CS 6501: Constrained-Aware Generative AI

Lecture Notes 9
Discrete Diffusion Models

Masked diffusion for tokens, joint control, and constraint injection without duality

Prof. Ferdinando Fioretto
Department of Computer Science, University of Virginia

Tuesday, February 10, 2026

Abstract

Discrete diffusion models extend the diffusion pattern of Lectures 7–8 to *categorical* objects such as token sequences, code, or molecular strings. The key shift is that the state space is not $\mathbb{R}^d$ but a finite vocabulary $\mathcal{V}$, so the forward "noising" process is a Markov chain over categorical variables rather than Gaussian perturbations. This matters for constrained-aware generation: (i) the model defines and samples from a *joint* distribution over all tokens, which weakens the sequential bottleneck and exposure bias of autoregressive decoding from Lecture 4; (ii) intermediate states are partially corrupted or masked, which creates natural insertion points for symbolic constraints, verifiers, and feasibility restorers; (iii) the model outputs live in probability simplices, so some constraints can be enforced directly by projection or proximal steps on distributions, reusing the optimization primitives from Lecture 6.

These notes introduce the D3PM formulation [Austin et al., 2021] and masked diffusion language modeling [MDLM, 2024], contrast them with ratio/score viewpoints such as SEDD [Lou et al., 2023], and then develop three practical mechanisms for injecting constraints: constrained unmasking policies, constrained decoding on intermediate representations, and simplex projection with constraints. The lecture is designed as the bridge into the course lab phase on *discrete* constraints, before we introduce duality-based methods.

1 Why discrete diffusion for constraint-aware generation?

Lecture 4 framed autoregressive (AR) generation as incremental search over prefixes, with natural insertion points for constraints that are *prefix-checkable* (finite-state, grammar, lexical). This framework is operational and widely used, but it exposes two structural frictions that become sharp in constrained settings.

First, AR sampling is sequential: each token must be generated before the next token distribution exists. When constraints are *global* (template completion, balanced structures, required sets of entities, program type correctness, molecule validity, safety policies that depend on the entire output), enforcing them through local token-by-token decisions is brittle, and search procedures can easily prune all feasible hypotheses early.

Second, the AR factorization controls the joint distribution indirectly by specifying conditionals $p_\theta(y_t \mid y_{<t}, x)$. Even if every local step is plausible, global validity can fail, and repairing a late error may require rewriting earlier tokens, which is awkward for prefix-based decoding.

Discrete diffusion offers a different interface: it defines a *joint* distribution over the full sequence and generates by iteratively denoising a corrupted sequence. At step t, many positions can change, and the model can revise earlier mistakes without rewinding a prefix. This makes constraint injection look closer to the "one model step, one constraint step" template from Lectures 6–7, except the object being updated is discrete.

Throughout we reuse the course's constrained target distribution viewpoint from Lecture 1. If $y \in \mathcal{V}^T$ is the output sequence and x is context, we want to sample from

$$\pi(y \mid x) \;\propto\; p_\theta(y \mid x) \exp(-\lambda\phi(y,x))\, \mathbf{1}\{y \in \mathcal{C}(x)\}, \tag{1}$$

where p_θ is a base generative model (here a discrete diffusion model), ϕ is a soft constraint potential (a verifier score, toxicity penalty, semantic alignment loss), and $\mathcal{C}(x)$ is a hard feasibility set (grammar, automaton, structured fields, domain rules). The lecture goal is to show how discrete diffusion creates practical insertion points for approximating inference under (1).

2 Discrete state space and probability simplices

Let $\mathcal{V}$ be a finite vocabulary of size $K := |\mathcal{V}|$. A categorical variable $z \in \mathcal{V}$ has a distribution $p \in \Delta(\mathcal{V}) := \{p \in \mathbb{R}^K_{\geq 0} : \sum_{v\in\mathcal{V}} p(v) = 1\}$. A length-$T$ token sequence is $y = (y_1, \ldots, y_T) \in \mathcal{V}^T$, and a factorized distribution across positions can be represented as a product of per-position simplices, $\prod_{i=1}^T \Delta(\mathcal{V})$. Importantly, a diffusion model's intermediate representation often includes such per-position distributions even when the eventual sample is discrete.

Two concrete representations will recur.

One-hot representation. For a token $z \in \mathcal{V}$, let $e(z) \in \{0,1\}^K$ be its one-hot vector. Then a categorical distribution is a convex combination of one-hots, and many diffusion kernels can be expressed as linear maps acting on one-hots.

Masked representation. Many language-oriented discrete diffusion models introduce a special absorbing symbol `[MASK]` and treat intermediate states as *partially observed*: some positions are actual tokens, some are `[MASK]`. This yields a state space $(\mathcal{V} \cup \{\texttt{[MASK]}\})^T$ and emphasizes the denoising-as-filling-in interpretation.

The simplex viewpoint is central for constraint-aware generation: several constraints can be expressed as linear constraints on probabilities (forbidden tokens, minimum mass on a set, marginals, and mixture constraints), enabling feasibility restoration by projection or proximal operators, exactly as in Lecture 6.

3 D3PM: structured denoising diffusion in discrete state-spaces

We present the D3PM formulation of Austin et al. [2021], which generalizes the continuous DDPM pattern from Lecture 7 to categorical states.

3.1 Forward process: categorical noising as a Markov chain

Fix a time horizon T and define a forward Markov chain $(y_t)_{t=0}^T$ on $\mathcal{V}^T$. For clarity, consider a single position first. Let $z_t \in \mathcal{V}$ denote the token at that position at time t. The forward noising is specified by a sequence of transition matrices $(Q_t)_{t=1}^T$ where each $Q_t \in \mathbb{R}^{K\times K}$ is row-stochastic:

$$q(z_t = v \mid z_{t-1} = u) = Q_t[u,v]. \tag{2}$$

Typical choices include "uniform corruption" kernels that slowly mix toward the uniform distribution, and "absorbing" kernels that gradually replace tokens by `[MASK]` (or by a generic noise token). For sequences, one often applies (2) independently across positions, yielding

$$q(y_t \mid y_{t-1}) = \prod_{i=1}^T q(y_{t,i} \mid y_{t-1,i})\,. \tag{3}$$

The cumulative transition matrix is $\bar{Q}_t := Q_1 Q_2 \cdots Q_t$, so the marginal is tractable:

$$q(z_t = v \mid z_0 = u) = \bar{Q}_t[u, v]. \tag{4}$$

This exact marginal tractability plays the role that the Gaussian closed form $q(x_t \mid x_0)$ played in Lecture 7.

3.2 Reverse model and training objective

We learn a reverse-time chain $p_\theta(z_{t-1} \mid z_t)$ (or $p_\theta(y_{t-1} \mid y_t, x)$ in the conditional case). A standard parameterization is to output logits over $\mathcal{V}$ and form a categorical distribution via softmax:

$$p_\theta(z_{t-1} = u \mid z_t = v, x) = \mathrm{softmax}(\ell_\theta(v, t, x))\,[u], \tag{5}$$

where ℓ_θ is produced by a Transformer-like network that takes the corrupted sequence at time t and optional conditioning x.

Training mirrors the DDPM variational bound: one can write a bound on $-\log p_\theta(y_0 \mid x)$ using the forward chain as an approximate posterior over intermediate states. The key point for practice is that the bound decomposes into a sum of KL terms between tractable categorical distributions and the learned reverse conditionals. This yields objectives that resemble cross-entropy on tokens at different noise levels.

In many implementations, rather than predicting a continuous noise ϵ, the model predicts either the original token y_0 (a denoising classifier) or the previous token y_{t-1}. The denoising classification objective has the form

$$\mathcal{L}_{\text{denoise}}(\theta) = \mathbb{E}_{t,\, y_0 \sim p_{\text{data}},\, y_t \sim q(\cdot \mid y_0)} \left[\, -\log p_\theta(y_0 \mid y_t, t, x)\, \right], \tag{6}$$

where $p_\theta(y_0 \mid y_t, t, x)$ is typically factorized across positions but shares parameters through the network. This objective is the discrete analogue of the "simple" objective in Lecture 7: it regresses to the clean object from a corrupted view.

Remark 1 (What changes relative to continuous DDPM)**.** *The computational pattern is the same: sample a data point, sample a time, corrupt it with the forward kernel, then train a network to predict a reverse object. The mathematical difference is that the noise model is* categorical*, so the score/gradient view must be replaced by either (i) ratio estimators (SEDD) or (ii) masking and classification (MDLM), and feasibility restoration can often be implemented directly as masking or projection in the simplex.*

4 Masked diffusion language models (MDLM) as an absorbing-kernel case

Masked diffusion is especially convenient for text because it matches the inductive bias of modern masked language modeling while still defining a diffusion-style *reverse* process. The key modeling move is to enlarge the state space from tokens to *tokens plus a mask symbol.* Let $\mathcal{V}$ be a vocabulary of size K and define $\mathcal{V}' := \mathcal{V} \cup \{\texttt{[MASK]}\}$. A sequence at time t is $y_t \in (\mathcal{V}')^T$, where masked positions represent uncertainty that will be resolved gradually.

4.1 Forward process: progressively masking with an absorbing state

A simple and widely used forward noising process replaces tokens by `[MASK]` with probability β_t at step t, and leaves them unchanged with probability $1 - \beta_t$. Once a position becomes masked, it stays masked forever under the forward process (the mask is absorbing). For a single variable $z_{t-1} \in \mathcal{V}'$, define

$$q(z_t \mid z_{t-1}) = \begin{cases} 1-\beta_t & \text{if } z_t = z_{t-1} \in \mathcal{V}, \\ \beta_t & \text{if } z_t = \texttt{[MASK]} \text{ and } z_{t-1} \in \mathcal{V}, \\ 1 & \text{if } z_{t-1} = \texttt{[MASK]} \text{ and } z_t = \texttt{[MASK]}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

For sequences $y_t \in (\mathcal{V}')^T$, one usually applies this independently across positions:

$$q(y_t \mid y_{t-1}) = \prod_{i=1}^{T} q(y_{t,i} \mid y_{t-1,i}). \tag{8}$$

This factorized forward kernel is easy to sample and yields a clean interpretation: as t increases, more positions become masked. The schedule (β_t) controls how rapidly information is destroyed.

Even though (7) is simple, it shares a structural property with the Gaussian forward process in continuous diffusion: we can compute marginals $q(y_t \mid y_0)$ efficiently because the corruption acts independently per position and has a closed-form description. In particular, for each position i, there is a time-dependent probability that it is still unmasked. Let

$$\alpha_t := \prod_{s=1}^{t} (1-\beta_s), \tag{9}$$

so α_t is the probability that a token survives (stays unmasked) through t forward steps. Then for any original token $y_{0,i} \in \mathcal{V}$,

$$q(y_{t,i} = y_{0,i} \mid y_{0,i}) = \alpha_t, \qquad q(y_{t,i} = \texttt{[MASK]} \mid y_{0,i}) = 1-\alpha_t. \tag{10}$$

Equation (10) is the masked analogue of the DDPM marginal $q(x_t \mid x_0)$ from Lecture 7: it lets us train by sampling a time t, corrupting y_0 to y_t, and then predicting some target based on (y_t, t).

4.2 Reverse-time modeling: "fill the masks" as iterative refinement

The reverse process conceptually undoes masking: it turns a heavily masked y_T into an unmasked $y_0 \in \mathcal{V}^T$. Unlike AR decoding, reverse steps can revise multiple positions because the model conditions on the full partially observed sequence.

There are two common modeling interfaces, and it is useful to keep them distinct.

(A) Predict the clean tokens (denoise-to-data). Train a network to predict the original tokens at masked (and possibly unmasked) positions:

$$p_\theta(y_0 \mid y_t, t, x) = \prod_{i=1}^{T} p_{\theta,t,i}(y_{0,i} \mid y_t, x), \tag{11}$$

where $p_{\theta,t,i}(\cdot)$ is categorical on $\mathcal{V}$ (often produced by logits and softmax). A simple objective is cross-entropy on the original tokens, evaluated on positions that were corrupted:

$$\mathcal{L}_{\text{MDLM}}(\theta) = \mathbb{E}_{t,\, y_0 \sim p_{\text{data}},\, y_t \sim q(\cdot \mid y_0)} \left[- \sum_{i: y_{t,i} = \texttt{[MASK]}} \log p_{\theta,t,i}(y_{0,i} \mid y_t, x) \right]. \tag{12}$$

At sampling time, a schedule chooses a subset of masked positions to fill using the distributions $p_{\theta,t,i}(\cdot)$, producing y_{t-1}.

(B) Predict reverse transitions. Alternatively, one can attempt to parameterize $p_\theta(y_{t-1} \mid y_t, t, x)$ directly, matching the D3PM style. In masked diffusion, this often reduces to specifying rules for how many masks are removed per step and how tokens are proposed.

For the constraint-aware agenda, both interfaces lead to the same practical insertion point: at each reverse step we obtain per-position distributions over candidate tokens for at least some positions. This is precisely what we will mask, project, or decode with constraints in Section 7.

4.3 Unmasking schedules and why they matter for constraints

A masked diffusion sampler must decide *when* each position becomes committed to a token. This decision is a schedule. A simple schedule fills a fixed fraction of remaining masks each step, or fills positions in random order. More structured schedules use uncertainty measures, such as selecting positions with lowest entropy first (easy tokens) or highest entropy first (hard tokens).

In constrained generation, the schedule is not merely a sampling detail. It is a control knob for feasibility. Many constraints are easier to satisfy when some positions remain flexible until the algorithm has stabilized the high-level structure. For example, if the output must satisfy a schema, it is often better to commit structural tokens (braces, commas, separators, required keys) before committing free-form values. If the output must contain a specific phrase, it can be helpful to reserve a masked span for that phrase until the surrounding context is coherent. These are instances of the general principle that constraints are often *structural* while quality is often *semantic*; masked diffusion lets us delay semantic commitment until structural feasibility is secured.

This is one of the central reasons that MDLM-style processes are attractive for the lab: they provide a direct mechanism (the unmasking schedule) to couple constraint progress with sampling dynamics without changing the base network.

4.4 Connection to AR modeling: joint distribution control versus conditional control

AR models define $p(y \mid x) = \prod_{i=1}^{T} p(y_i \mid y_{<i}, x)$ and are trained to predict each next token given a prefix. Constraints are enforced by manipulating decoding, but the model itself does not revise earlier tokens. Masked diffusion instead represents uncertainty with `[MASK]` and repeatedly refines a full configuration. The reverse chain therefore behaves like a joint optimizer over token configurations: it can repair earlier choices because those choices are not privileged as an irreversible prefix.

This is the concrete meaning of "joint distribution control versus AR conditionals" in this lecture. It does not mean that the model is magically globally optimal. It means the sampling interface gives you repeated opportunities to enforce constraints, repair violations, and reallocate probability mass, which is exactly the insertion-point story from Lectures 6–7 transported to discrete spaces.

4.5 A constraint-aware view of MDLM: feasibility on partial information

A final advantage of masked diffusion for constraints is that partial sequences can still be checked. Many constraints admit meaningful evaluation on incomplete objects. A parser can rule out illegal next tokens even when some positions are unknown. A lexical constraint tracker can maintain which required words are still missing. A schema checker can maintain which fields are already present. This yields an incremental feasibility state σ_t that is updated as masks are filled and that can restrict candidate tokens at each step. This will be formalized in Section 7 as constrained unmasking policies and constrained decoding over intermediate representations.

From the perspective of constrained-aware generative modeling, MDLM can therefore be summarized as follows: it replaces the AR notion of "prefix" with a notion of "partial observation," and this partial observation is often a better substrate for symbolic constraint tracking and targeted repair.

5 SEDD: ratio estimation as a discrete analogue of scores

In continuous diffusion (Lecture 7), the score $\nabla_x \log p_t(x)$ provides a local object that drives reverse-time dynamics and supports guidance by additive modifications of that score. In discrete spaces, there is no derivative with respect to tokens, so we need a different local object that plays an analogous role. One principled choice is to work with *log-probability ratios* between neighboring configurations. This is the organizing idea in SEDD [Lou et al., 2023], and it is also the right abstraction for constraint injection because many constraints act by selectively reweighting candidate tokens relative to one another.

5.1 From gradients to local log-ratios on neighborhoods

Let Ω be a discrete state space. For language we can take $\Omega = \mathcal{V}^T$, and for a single position $\Omega = \mathcal{V}$. A discrete analogue of a "local derivative" is obtained by fixing a neighborhood structure (a graph) on Ω. The simplest and most common choice for sequences is the Hamming-1 neighborhood: y' is a neighbor of y if the two sequences differ at exactly one position. For each state y and each neighbor y', define the log-ratio

$$\rho_t(y \to y') := \log \frac{p_t(y')}{p_t(y)}. \tag{13}$$

If we can evaluate $\rho_t(y \to y')$ for all neighbors, then we can define local moves that prefer higher-probability neighbors, exactly the discrete analogue of flowing uphill along $\nabla_x \log p_t(x)$.

For a single categorical variable $z \in \mathcal{V}$, a standard reduction fixes a reference token $v^\star$ and parameterizes the distribution via log-ratios

$$r_t(v) := \log \frac{p_t(z = v)}{p_t(z = v^\star)}. \tag{14}$$

Given $\{r_t(v)\}_{v\in\mathcal{V}}$, the distribution is recovered by normalization:

$$p_t(z = v) = \frac{\exp(r_t(v))}{\sum_{u\in\mathcal{V}} \exp(r_t(u))}, \qquad \text{with } r_t(v^\star) = 0. \tag{15}$$

This mirrors the continuous identity that knowing the score field determines the density up to normalization constants (under appropriate integrability conditions), but now the object is discrete and defined relative to a reference.

5.2 Why ratios are the right interface for guidance and constraints

Ratios are especially convenient because many modifications to a base distribution are multiplicative, and multiplicative changes become additive in log-ratio space. Suppose we define a constrained or guided target at time t by reweighting p_t using an energy (or penalty) term $U_t(y)$:

$$\tilde{p}_t(y) \;\propto\; p_t(y) \exp(-U_t(y)). \tag{16}$$

Then the log-ratio of $\tilde{p}_t$ between neighbors is

$$\log \frac{\tilde{p}_t(y')}{\tilde{p}_t(y)} = \log \frac{p_t(y')}{p_t(y)} - \big(U_t(y') - U_t(y)\big) = \rho_t(y \to y') - \Delta U_t(y \to y'). \tag{17}$$

Equation (17) is the discrete analogue of guided scores: you keep the model's local preference ρ_t, and you add an additive correction derived from the constraint potential. This is conceptually the same pattern as classifier guidance or energy guidance in continuous diffusion, but expressed in an object that is well-defined on token spaces.

In practice, constraints are often not expressed as smooth energies but as symbolic admissibility. Even then, ratios remain useful. If a token choice is forbidden, the corresponding move can be assigned log-ratio $-\infty$ (or equivalently probability zero after normalization). If a constraint requires allocating probability mass to a subset, the ratio correction can be interpreted as a log-barrier or soft preference that pushes probability mass toward that subset. This viewpoint will be useful later when we discuss the boundary between hard feasibility restoration (masking, projection) and soft preference shaping (guidance).

5.3 SEDD at a high level

SEDD [Lou et al., 2023] builds a diffusion-like generative process on discrete spaces using ratio estimation rather than direct denoising classification. The model learns to predict suitable local ratio objects that permit reverse-time sampling over discrete states. The key takeaway for this course is not the full derivation, but the interface it provides: rather than a per-position "predict the clean token" objective, ratio-based methods emphasize estimators that support local moves and that can be straightforwardly modified by additive constraint corrections, as in (17).

This makes SEDD a natural conceptual bridge between (i) score-based continuous diffusion and (ii) constrained discrete generation where the most actionable signals are local comparisons between token alternatives under a partially specified structure.

6 Sampling: ancestral updates, parallelism, and blocks

Sampling is where discrete diffusion differs most from AR decoding in an operational sense. Even when the learned model resembles a masked language model producing logits, the sampling procedure is not a single left-to-right pass. Instead it is a reverse-time chain that repeatedly revises a partially specified sequence. This section makes explicit what is being sampled and why different update granularities (single-site, subset, block) matter for constraints.

6.1 What a reverse step produces

At reverse time t, the sampler has a partially corrupted configuration y_t. A reverse model can be parameterized in (at least) two common ways.

The first is *direct reverse transition prediction*: a network parameterizes $p_\theta(y_{t-1} \mid y_t, t, x)$, often factorized across positions or blocks conditioned on the full context. This is closer to the original D3PM variational formulation [Austin et al., 2021].

The second is *denoise-to-data prediction*: a network predicts a distribution over the clean data, $p_\theta(y_0 \mid y_t, t, x)$, and a reverse step is implemented by sampling a partially denoised state from this prediction with a schedule that determines which positions are committed at each time. Masked diffusion language models frequently use this interface [MDLM, 2024] because it is natural to treat y_t as a masked sequence and predict tokens for the masked positions.

Both parameterizations yield, at the level of implementation, per-position (or per-block) categorical distributions. This is exactly what we need to insert constraints: we can mask distributions, project them onto constrained simplices, or run a constrained inner decoding routine without changing the network.

6.2 Single-site versus subset updates

A convenient mental model for discrete reverse-time sampling is *iterated local updates*. At each step, the sampler chooses a subset of positions $m_t \subseteq \{1, \ldots, T\}$ to update and leaves the rest unchanged. When m_t is a singleton, this resembles Glauber dynamics on sequences. When m_t includes many positions, the step becomes a partially parallel update.

Subset updates matter for constraints because they expose a compute allocation mechanism. If the constraint checker identifies a small set of positions responsible for a violation (for example, a mismatched bracket or a missing required key), then choosing m_t to focus on those positions yields an efficient repair-oriented sampler. Conversely, if we update all positions aggressively, we often introduce unnecessary randomness that destabilizes already-feasible structure. The constrained-aware sampling agenda therefore benefits from update schedules that become more conservative over time and that focus on violating regions.

6.3 Parallelism and the "joint control" advantage

Discrete diffusion is sometimes summarized as providing a "joint" model rather than an AR factorization. Operationally, the advantage is that the model conditions on the entire current configuration y_t and can propose revisions to multiple positions based on global context. Even though the Markov chain runs over time, the per-step computation is parallel over positions, and early choices do not become irrevocable prefixes.

This matters for global constraints. If a constraint couples distant positions, AR decoding must maintain many prefixes to avoid premature commitment. Diffusion-style revision offers a different degree of freedom: you keep a single configuration but revisit it multiple times. Constraints can therefore act repeatedly, shaping the configuration toward feasibility without requiring large beams.

6.4 Blocks as a middle ground between AR and fully-parallel updates

Block diffusion [Arriola et al., 2025] formalizes a continuum between AR generation and diffusion by controlling the block size updated per step. When the block is of size one and chosen as the next unfilled position, the procedure resembles AR sampling. When blocks cover many positions, the procedure resembles masked diffusion with parallel fill.

Block updates are particularly well-matched to constraints that have local structural scope. For instance, a grammar constraint often couples a short span around a delimiter, and a lexical constraint may couple a short phrase. Updating that span as a block can be more stable than updating tokens independently because it respects the coupled nature of the constraint. This is the same intuition behind structured prediction: coupled decisions are better handled jointly than via independent marginals.

6.5 Argmax flows and categorical diffusion variants

Beyond D3PM and masked diffusion, categorical diffusion has other parameterizations that emphasize different sampling primitives. Multinomial diffusion and argmax flows [Hoogeboom et al., 2021] view the reverse process as operating on categorical distributions that are later mapped to discrete tokens by an argmax-type operation. While the technical details differ, the constrained-aware relevance is consistent: many of these variants expose a distributional intermediate representation at each step. Once we have that distributional representation, we can insert feasibility restoration operators, either as admissible-set masking or as simplex projections, before sampling a discrete token.

This perspective is useful for the lab: students can work at the level of logits and simplex operations, even if the underlying paper uses a different formalism for categorical diffusion.

6.6 Practical knobs that interact with constraints

Three practical design choices often matter for constrained diffusion in discrete spaces.

First, the *noise schedule* (or masking schedule) controls how quickly the process commits to discrete tokens. A schedule that commits too fast tends to freeze constraint violations, while a schedule that commits too slowly can waste compute by repeatedly revisiting unconstrained regions. Constraint-aware schedules often become aggressive only after the constraint state suggests that global feasibility is within reach.

Second, the *sampling temperature* and related smoothing choices determine how exploratory the chain is. Higher exploration can help escape local traps, but it also increases the rate at which feasibility is broken and must be repaired. In constrained settings, it is common to combine exploration early with stricter feasibility restoration late.

Third, the *update policy* (single-site, subset, block) interacts with the constraint checker. If the checker can localize violations, then targeted updates become a direct method for allocating the reverse steps to the most informative repairs. This is one of the most important algorithmic reasons that diffusion-style iterative refinement is attractive for discrete constraints.

These knobs will be important in the lab phase because they determine the computational cost of constraint satisfaction and the quality-validity tradeoff observed in practice.

7 Constraint injection mechanisms for discrete diffusion

We now develop three mechanisms for enforcing constraints within discrete diffusion sampling. The goal is to keep the mechanisms algorithmic and modular, mirroring the explicit insertion points we emphasized in Lectures 4, 6, and 7.

7.1 Mechanism (i): constrained unmasking as a control policy

In masked diffusion, a step consists of selecting which masked positions to reveal (or which positions to resample). This selection is itself a *policy.* In unconstrained sampling the policy is typically fixed (uniform random, schedule-based, or entropy-based). In constrained-aware sampling, the policy can be adapted to the constraint state.

Let $m_t \subseteq \{1, \dots, T\}$ denote the set of positions that will be *updated* at step t. We can view m_t as the action of a controller:

$$m_t \sim \pi_{\text{unmask}}(\cdot \mid y_t, \sigma_t), \tag{18}$$

where σ_t summarizes constraint progress (for example, an automaton state, a set of unsatisfied lexical items, or a parser configuration).

The key idea is that constraints often suggest *where* to allocate modeling capacity. If a constraint requires inserting a specific phrase, the policy should focus updates near plausible insertion sites. If a grammar constraint is violated by the current partial structure, the policy should prioritize positions that can repair the violation. This resembles resource allocation in constrained beam search (Lecture 4), except the resource is "which positions to commit now" rather than "which prefixes to keep alive".

Example 1 (Lexical constraints as unmasking control)**.** *Suppose the output must include a required phrase, such as a chemical functional group name or a specific field in a JSON object. Maintain a constraint state σ_t that tracks whether the phrase has been completed. If not, identify a window of positions where the phrase can be inserted and prioritize those positions in m_t. In practice, this can be implemented by (i) reserving a contiguous masked span for the phrase and (ii) unmasking it with a shorter schedule than the rest. This biases sampling toward satisfying the hard requirement early, reducing the risk that later steps cannot accommodate it.*

Remark 2 (Why this is "constraints without duality")**.** *The unmasking policy changes* which variables are updated*, not the model objective. It is a control knob that can enforce hard constraints by preserving degrees of freedom until constraint-critical content is inserted, a purely algorithmic mechanism that does not require Lagrange multipliers or dual updates.*

7.2 Mechanism (ii): constrained decoding on intermediate representations

Even though the underlying model is not AR, the reverse step typically produces per-position distributions (logits) over tokens. This yields an intermediate *decoding* problem: decide which token to assign to each updated position, given the current corrupted context.

The simplest constraint insertion is to reuse the constrained decoding tools of Lecture 4 locally. Consider a hard constraint represented by an automaton or grammar. At time t, we have a partially specified sequence y_t with masked positions. We can maintain a constraint state σ_t that depends on the visible tokens and is updated as we fill masked positions.

Operationally, for each position $i \in m_t$ we obtain a distribution $p_{\theta,t,i}(\cdot) \in \Delta(\mathcal{V})$ and then apply a constraint mask:

$$\tilde{p}_{\theta,t,i}(v) \;\propto\; p_{\theta,t,i}(v)\,\mathbf{1}\{v \in A(\sigma_t, i)\}, \tag{19}$$

where $A(\sigma_t, i)$ is the set of tokens admissible at position i given the current constraint state. This is the direct analogue of automaton masking in Lecture 4, except it is applied at *intermediate* steps and potentially at multiple positions in parallel.

The nontrivial part is that constraints couple positions. Two standard patterns are:

Sequential within-step filling. Even if m_t includes many positions, fill them one by one in an order, updating σ_t after each fill, using (19). This recovers constrained decoding behavior while preserving the diffusion outer loop.

Beam or search over the block. For a block m_t (as in block diffusion), treat the block filling as a small structured prediction problem and run a constrained search within the block (beam search, dynamic beam allocation, or a grammar parser) while keeping the outside positions fixed. This is attractive when constraints couple a local contiguous region.

Example 2 (JSON schema as intermediate decoding)**.** *Suppose we must generate a JSON object with required keys and well-formed syntax. Maintain a parser state σ_t for the JSON grammar (or a compiled token-level automaton). At a diffusion step, when filling a masked position that lies inside the JSON structure, compute admissible tokens $A(\sigma_t, i)$ from the parser, mask logits by* (19)*, then sample or select. The diffusion model supplies semantic plausibility (values, wording) while the parser guarantees syntactic validity.*

This mechanism is the discrete analogue of "projection after each reverse update" in Lecture 7: it enforces feasibility at each step, but feasibility is checked in a symbolic representation rather than as a Euclidean distance.

7.3 Mechanism (iii): projection onto probability simplices with constraints

The previous mechanism enforces constraints by *masking* and renormalizing. This is equivalent to a projection onto a face of the simplex (zeroing forbidden tokens). More generally, we can enforce constraints that are naturally expressed as *linear* or *convex* constraints on distributions by solving a projection or proximal problem, using the primitives from Lecture 6.

Consider a per-position distribution $p \in \Delta(\mathcal{V})$ produced by the model (for some position i at some time t). Let $\mathcal{C}_\Delta \subseteq \Delta(\mathcal{V})$ be a convex constraint set over distributions, for example

$$\mathcal{C}_\Delta = \{p \in \Delta(\mathcal{V}) : p(v) = 0 \;\forall v \in \mathcal{F},\; \sum_{v \in \mathcal{S}} p(v) \geq \rho\}, \tag{20}$$

where $\mathcal{F}$ is a set of forbidden tokens and $\mathcal{S}$ is a set of required tokens that must receive at least ρ total probability mass.

Given a base distribution p, define a feasibility restoration operator

$$R(p) := \mathrm{Proj}_{\mathcal{C}_\Delta}(p) = \arg\min_{q \in \mathcal{C}_\Delta} \|q - p\|_2^2. \tag{21}$$

This is a convex problem. The resulting $q = R(p)$ is the closest distribution that satisfies the constraints, and it can be used as the sampling distribution for that position.

Two observations connect this to the course's earlier material.

First, (21) is an explicit projection, exactly as in Lecture 6, except the ambient space is the simplex. Many such projections admit fast algorithms (sorting-based simplex projection; projections with box constraints; projections with a small number of linear inequalities).

Second, projection can be used even for constraints that are not hard feasibility in token space but rather *risk control* in probability space: one can ensure that the probability of unsafe tokens is bounded, or that the model maintains a minimum probability mass on "escape" options that can complete a grammar.

Example 3 (Soft lexical pressure via simplex constraints)**.** *Suppose a phrase must appear somewhere, but we treat it as a soft constraint. In a diffusion model, we can allocate a set of candidate positions where the phrase could begin. For each such position, define $\mathcal{S}$ as the set of tokens that are valid phrase starters and enforce $\sum_{v \in \mathcal{S}} p(v) \geq \rho_t$ for a time-dependent schedule ρ_t that increases as t decreases. This can be implemented by projecting per-position distributions using* (21)*. Unlike hard masking, this preserves flexibility and can be tuned by ρ_t.*

Remark 3 (Relation to guidance for discrete diffusion)**.** *Guidance in continuous diffusion adds an extra score term to shift probabilities. Simplex projection can be seen as a complementary mechanism: rather than modifying logits by additive energies, it enforces feasibility by moving the resulting distribution to a constrained set. In practice, one often combines both: add an energy shaping term to logits (soft constraint) and then project to enforce hard safety or syntax constraints.*

8 A generic sampler with explicit constraint insertion points

We now write a generic discrete diffusion sampler that makes the insertion points explicit. It mirrors Algorithm 1 in Lecture 7, but replaces "mean + Gaussian noise" with "categorical sampling with optional masked positions", and replaces Euclidean projection with symbolic masking or simplex projection.

Algorithm 1 Discrete diffusion sampling with constraint insertions (generic template)

```
Require: time horizon $T$; reverse model $p_\theta(\cdot \mid y_t, t, x)$; unmask policy $\pi_{\text{unmask}}$; constraint state update rule; restoration operator $R$ (masking or simplex projection); optional guidance operator $G$.
Ensure: sample $y_0$.
1: Initialize $y_T \sim q_T$ (e.g., all [MASK] or a high-noise categorical sample); initialize constraint state $\sigma_T$.
2: for $t = T, T-1, \dots, 1$ do
3:     Choose update set $m_t \sim \pi_{\text{unmask}}(\cdot \mid y_t, \sigma_t)$.
4:     Compute model distributions $\{p_{\theta,t,i}(\cdot)\}_{i \in m_t}$.
5:     Optionally apply guidance: $p_{\theta,t,i} \leftarrow G(p_{\theta,t,i}, y_t, \sigma_t, t)$.
6:     Restore feasibility in distribution space: $q_{t,i} \leftarrow R(p_{\theta,t,i}; \sigma_t, i)$ for each $i \in m_t$.
7:     Sample or select tokens for $i \in m_t$ from $q_{t,i}$ to produce $y_{t-1}$ (positions not in $m_t$ copy from $y_t$).
8:     Update constraint state $\sigma_{t-1}$ from $(y_{t-1}, \sigma_t)$.
9: end for
10: return $y_0$.
```

Algorithm 1 clarifies how the three mechanisms fit together. Mechanism (i) is π_{unmask}. Mechanism (ii) is R implemented as symbolic admissible-set masking, and mechanism (iii) is

R implemented as a projection/prox step on simplices. This template also anticipates the constrained discrete diffusion approach in Cardei et al. [2025].

9 Worked examples

We close with three examples that connect directly to the constraint taxonomy and insertion patterns in earlier lectures.

9.1 Example 1: lexically constrained generation beyond beam search

Lecture 4 presented dynamic beam allocation [Post and Vilar, 2018] for enforcing lexical constraints in AR generation. Discrete diffusion provides an alternative: rather than allocating beam budget across constraint states, we allocate *update budget across positions* and enforce lexical progress through unmasking control and intermediate constrained decoding.

If the required phrase is a short sequence of tokens, we can reserve a masked span and treat its filling as a block update, running a small constrained search inside the block while the rest of the sequence is held fixed. This makes constraint satisfaction robust even when the base model assigns low probability to completions that contain the phrase, because we prevent early commitment elsewhere from consuming the length budget.

9.2 Example 2: grammar constraints for code or structured text

For code or formal languages, constraints are often context-free (balanced delimiters, typing). AR constrained decoding enforces these constraints locally but is still sensitive to early mistakes. In masked diffusion, the model can revise multiple positions and can fill structural tokens (brackets, commas, keywords) before details. A practical policy is to unmask structural tokens early, updating the parser state, and unmask identifiers and literals later. This is a discrete analogue of "coarse-to-fine" constraint satisfaction.

9.3 Example 3: SMILES validity and domain rules

SMILES strings must satisfy syntax constraints and chemical validity constraints. Grammar constraints can ensure syntactic validity, while chemical validity may require additional checks (valency, ring closure consistency). In diffusion sampling, one can (i) enforce SMILES grammar by masking logits with an automaton; (ii) incorporate a chemical validity checker as a repair operator that suggests which positions to resample (unmask policy); and (iii) enforce simple safety constraints in simplex space by banning problematic tokens or motifs as forbidden sets $\mathcal{F}$ in (20). This combination produces a modular constrained-generation loop with clear insertion points for domain tools.

10 Discussion: guidance, risk, and evaluation in discrete diffusion

Several guidance mechanisms proposed for discrete diffusion can be understood as modifying logits (or ratios) by additive terms derived from a discriminator, a reward model, or a verifier, in the same spirit as guidance in Lecture 7 but adapted to categorical distributions [Schiff et al., 2025]. In constrained settings, it is often safer to treat guidance as *soft* and to pair it with hard feasibility restoration (symbolic masking or simplex projection), because strong guidance can still produce fluent but invalid sequences if it is not tied to an executable constraint.

Evaluation in discrete constrained generation should separate three axes. Validity measures whether hard constraints are satisfied (grammar acceptance, schema compliance, presence of required items). Violation magnitude measures how badly constraints fail when they fail (useful for soft constraints and partial validity). Quality and diversity should be measured in a task-appropriate way (human evaluation, reference-based metrics when appropriate, or domain scores), but, as emphasized in Lecture 1, these are not substitutes for validity.

11 Summary and looking ahead

Discrete diffusion models provide an iterative, joint-generation interface for token sequences, complementing autoregressive decoding. D3PM formalizes categorical noising and denoising via Markov chains with tractable marginals, while MDLM emphasizes masking as a practical corruption mechanism. Ratio-based viewpoints such as SEDD offer a discrete analogue of scores and connect naturally to guidance as additive preference shaping.

From the constrained-aware perspective, the central benefit is algorithmic: the reverse process offers explicit insertion points. Constraints can act as (i) *constrained unmasking policies* that allocate update budget to constraint-critical positions, (ii) *constrained decoding* procedures applied to intermediate distributions using automata or grammars, and (iii) *simplex projections* that enforce probabilistic constraints directly on the model's output distributions. These mechanisms set up the lab phase on discrete constraints while staying within the optimization primitives of Lecture 6 and the iterative control pattern of Lectures 7–8, without invoking duality yet.

References

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Aaron van den Oord. Structured denoising diffusion models in discrete state-spaces (D3PM). In *NeurIPS*, 2021.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution (SEDD). *arXiv preprint arXiv:2310.16834*, 2023.

Emiel Hoogeboom, Didrik Nielsen, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. In *NeurIPS*, 2021.

Simple and effective masked diffusion language models (MDLM). *arXiv preprint arXiv:2406.07524*, 2024.

Matt Post and David Vilar. Fast lexically constrained decoding with dynamic beam allocation. In *NAACL*, 2018.

Michael Cardei, Ferdinando Fioretto, and collaborators. Constrained language generation with discrete diffusion models. *arXiv preprint arXiv:2503.09790*, 2025.

Yair Schiff and collaborators. Simple guidance mechanisms for discrete diffusion models. In *ICLR*, 2025.

Pablo Arriola and collaborators. Block diffusion: Interpolating between autoregressive and diffusion language models. *arXiv preprint arXiv:2503.09573*, 2025.