CS 6501: Constrained-Aware Generative AI

Lecture Notes 4
Autoregressive Transformers and Decoding

Generation as search under probabilistic and symbolic constraints

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

Thursday, January 22, 2026

Abstract

Autoregressive Transformers are the dominant foundation for modern language models because they couple a tractable likelihood with a powerful sequence architecture. In constrained-aware generation, this matters for two reasons. First, the factorization yields a canonical probabilistic objective, so "validity" can be expressed as a distributional constraint that modifies a base model. Second, inference at generation time is an explicit *search* over token sequences, so constraints can be injected directly into decoding by pruning invalid continuations and reweighting feasible candidates. These notes review the autoregressive factorization, the Transformer architecture, and standard decoding methods, then introduce constrained decoding via grammars and finite-state constraints as the first concrete instance of "generation as search subject to constraints".

1 From constrained generation to constrained decoding

Lecture 1 introduced a recurring template for constrained-aware generation: starting from a base model $p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{c})$ that encodes plausibility, and combining it with soft penalties and hard feasibility requirements to define a constrained target distribution $\pi(\mathbf{x} \mid \mathbf{c})$. Autoregressive language models specialize this view to sequences $\mathbf{y} = (y_1, \ldots, y_T)$ over a vocabulary $\mathcal{V}$, conditioned on a prompt or context $\mathbf{x}$ (which may itself be a token sequence). The central algorithmic fact for this lecture is that, for autoregressive models, *generation is an incremental search over prefixes.* This opens a direct insertion point for constraints that act on partial sequences.

Constrained decoding as constrained inference

Given a prompt $\mathbf{x}$, an autoregressive model defines $p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$. A hard constraint set $\mathcal{C}(\mathbf{x}) \subseteq \mathcal{V}^T$ induces the constrained distribution

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

and a soft constraint $\phi(\mathbf{y}, \mathbf{x})$ yields the energy-shaped target

$$\pi(\mathbf{y} \mid \mathbf{x}) \;\propto\; p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})\, \exp(-\lambda\, \phi(\mathbf{y}, \mathbf{x}))\,. \tag{2}$$

Decoding is any procedure that approximates either sampling $\mathbf{y} \sim \pi(\cdot \mid \mathbf{x})$ or optimization $\hat{\mathbf{y}} \in \arg\max_{\mathbf{y}} \log \pi(\mathbf{y} \mid \mathbf{x})$.

The remainder of the lecture makes the above concrete by (i) recalling how $p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$ is defined and trained for Transformers, (ii) formalizing standard decoding rules, and (iii) showing how grammars and finite-state constraints can be enforced at the level of prefixes.

2 Autoregressive factorization and maximum likelihood

Let $\mathbf{y} = (y_1, \ldots, y_T)$ be a token sequence. The autoregressive assumption models the joint distribution as

$$p_{\boldsymbol{\theta}}(\mathbf{y}) \;=\; \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(y_t \mid y_{<t}), \qquad \text{where} \qquad y_{<t} := (y_1, \ldots, y_{t-1}). \tag{3}$$

In conditional form, given a prompt $\mathbf{x}$, the factorization becomes

$$p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}) \;=\; \prod_{t=1}^{T} p_{\boldsymbol{\theta}}(y_t \mid y_{<t}, \mathbf{x}). \tag{4}$$

Training by maximum likelihood on a dataset $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$ corresponds to minimizing the negative log-likelihood, equivalently the token-level cross-entropy:

$$\min_{\boldsymbol{\theta}} \; \mathcal{L}(\boldsymbol{\theta}) \;:=\; -\sum_{i=1}^{n}\sum_{t=1}^{T_i} \log p_{\boldsymbol{\theta}}\left(y_t^{(i)} \mid y_{<t}^{(i)}, \mathbf{x}^{(i)}\right). \tag{5}$$

Here T_i counts predicted tokens up to (and including) `EOS` but excludes padding; in a decoder-only setting the prompt $\mathbf{x}$ is typically concatenated ahead of $\mathbf{y}$ so that $y_{<t}$ includes the prompt tokens. This objective is optimized with teacher forcing: during training, the model conditions on the *true* prefix $y_{<t}^{(i)}$. At test time, the model conditions on its own generated prefix, which yields exposure bias and compounds errors [Bengio et al., 2015, Ranzato et al., 2016]. From the viewpoint of constrained-aware generation, this gap is important: if constraints are not enforced during decoding, small local errors can push the model into regions where global validity is hard to recover.

A standard evaluation summary of (5) is perplexity. For a test set, define the average per-token negative log-likelihood ℓ as (5) divided by the total number of tokens. The perplexity is then $\mathrm{PPL} := \exp(\ell)$. Perplexity is useful as a calibration measure of probabilistic fit, but it is not a validity guarantee: a low-perplexity continuation may still violate grammar constraints, lexical requirements, or policy constraints [Theis et al., 2016, Holtzman et al., 2020].

3 Transformers for autoregressive modeling

The autoregressive factorization (4) reduces conditional generation to learning a family of next-token conditionals $p_{\boldsymbol{\theta}}(y_t \mid y_{<t}, \mathbf{x})$ that can be evaluated efficiently for all prefixes in parallel and updated as the prefix grows. At this point the modeling question is: which parameterization should we choose for these conditional distributions? In principle, many sequence models can serve this role, including n-gram, recurrent networks, and state-space models. However, in modern large-scale language modeling, the *Transformer* architecture has emerged as the dominant parameterization of $p_{\boldsymbol{\theta}}$ because it supports stable optimization at scale, exposes a highly expressive attention-based representation of the context $(y_{<t}, \mathbf{x})$, and can be implemented with hardware-efficient batched matrix operations [Vaswani et al., 2017]. We therefore focus on autoregressive Transformers as the canonical instantiation of (4), and then study decoding as the induced search problem for high-probability sequences, including the first constrained variants.

Token and position representations. Let an input sequence of length L be represented by token ids $(u_1, \ldots, u_L)$ with $u_i \in \mathcal{V}$, where $\mathcal{V}$ is a finite vocabulary and $|\mathcal{V}|$ denotes its size. A Transformer first maps each token id to a dense vector in $\mathbb{R}^d$ through an embedding table

$E \in \mathbb{R}^{|\mathcal{V}|\times d}$, so that $e_i := E[u_i] \in \mathbb{R}^d$. Because attention is permutation-invariant in its basic form, the model must also encode the position index $i \in \{1,\dots,L\}$. This is done by adding a positional encoding $p_i \in \mathbb{R}^d$, either learned (via a separate table $P \in \mathbb{R}^{L_{\max}\times d}$) or fixed (e.g., sinusoidal). The initial hidden state at position i is then

$$h_i^{(0)} := e_i + p_i \in \mathbb{R}^d.$$

Stacking the vectors row-wise yields a matrix of hidden states $H^{(0)} \in \mathbb{R}^{L\times d}$ whose i-th row is $h_i^{(0)\top}$.

Self-attention with a causal mask. The central operation in a Transformer layer is to update each position's representation by taking a content-dependent weighted average of representations at other positions. Fix a layer index $\ell \in \{1,\dots,L_{\text{layers}}\}$ and suppose the previous layer provides hidden states $H^{(\ell-1)} \in \mathbb{R}^{L\times d}$, where the i-th row $h_i^{(\ell-1)\top}$ summarizes the prefix information available to position i at depth $\ell-1$.

Self-attention is parameterized via three linear maps applied to every position: the *queries*, *keys*, and *values*. Concretely, choose a head dimension d_k and define learned matrices $W_Q, W_K, W_V \in \mathbb{R}^{d\times d_k}$. The projected matrices are

$$Q := H^{(\ell-1)}W_Q \in \mathbb{R}^{L\times d_k}, \qquad K := H^{(\ell-1)}W_K \in \mathbb{R}^{L\times d_k}, \qquad V := H^{(\ell-1)}W_V \in \mathbb{R}^{L\times d_k}.$$

Write $q_i^\top$ for the i-th row of Q and $k_j^\top$ for the j-th row of K. The scalar compatibility score between position i (as a query) and position j (as a key) is the dot product $\langle q_i, k_j\rangle = q_i^\top k_j$. Collecting all pairwise scores yields the matrix $S := QK^\top \in \mathbb{R}^{L\times L}$, where $S_{ij} = q_i^\top k_j$. The scaled dot-product attention operator is

$$\mathrm{Attn}(Q,K,V) \;=\; AV \qquad \text{with} \qquad A := \mathrm{softmax}\!\left(\frac{QK^\top}{\sqrt{d_k}} + M\right) \in \mathbb{R}^{L\times L}, \tag{6}$$

where $\sqrt{d_k}$ is a normalization that stabilizes magnitudes of dot products as d_k grows, and the softmax is applied row-wise, i.e., for each i,

$$A_{ij} \;=\; \frac{\exp\!\left(\frac{S_{ij}}{\sqrt{d_k}} + M_{ij}\right)}{\sum_{r=1}^{L}\exp\!\left(\frac{S_{ir}}{\sqrt{d_k}} + M_{ir}\right)}.$$

Thus, the updated representation at position i is a convex combination of value vectors: if $v_j^\top$ is the j-th row of V, then the i-th row of $\mathrm{Attn}(Q,K,V)$ equals $\sum_{j=1}^{L} A_{ij} v_j$.

For autoregressive language modeling, attention must be *causal*: position i is allowed to depend only on positions $j \le i$. This is enforced via an additive mask matrix $M \in \mathbb{R}^{L\times L}$ defined by

$$M_{ij} := \begin{cases} 0, & j \le i, \\ -\infty, & j > i, \end{cases}$$

so that after adding M and applying the softmax, all weights A_{ij} with $j > i$ become 0. In practice, $-\infty$ is implemented as a large negative constant, and padding positions can be masked by setting the corresponding entries in M to $-\infty$ as well, ensuring that attention ignores padded tokens.

Multi-head attention increases expressivity by running several attention operators in parallel with different learned projections. With H heads, one typically chooses $d_k = d/H$ so the total width remains d. For head $h \in \{1,\dots,H\}$, define $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)} \in \mathbb{R}^{d\times d_k}$, compute $\mathrm{Attn}^{(h)} \in \mathbb{R}^{L\times d_k}$ via (6), and then concatenate head outputs along the feature dimension:

$$\mathrm{MHA}(H^{(\ell-1)}) \;:=\; \left[\mathrm{Attn}^{(1)} \,\|\, \cdots \,\|\, \mathrm{Attn}^{(H)}\right] W_O,$$

where $\|$ denotes concatenation, the bracketed term lies in $\mathbb{R}^{L\times d}$, and $W_O \in \mathbb{R}^{d\times d}$ is an output projection. Intuitively, different heads can specialize to different relations, such as local syntactic dependencies, long-range coreference, or delimiter matching that is relevant for constrained generation.

A standard decoder block then combines attention with a position-wise feedforward network (FFN) and stabilization layers. A common choice is the two-layer FFN

$$\mathrm{FFN}(x) := W_2\,\sigma(W_1 x + b_1) + b_2,$$

applied independently to each position, where $W_1 \in \mathbb{R}^{d\times d_{\mathrm{ff}}}$, $W_2 \in \mathbb{R}^{d_{\mathrm{ff}}\times d}$, d_{ff} is an expansion dimension, and σ is a nonlinearity (e.g., GELU). Residual connections and normalization (LayerNorm or RMSNorm) are applied around both the attention and FFN sublayers; in the common pre-norm form,

$$\tilde{H} \;=\; H^{(\ell-1)} + \mathrm{MHA}(\mathrm{Norm}(H^{(\ell-1)})), \qquad H^{(\ell)} \;=\; \tilde{H} + \mathrm{FFN}(\mathrm{Norm}(\tilde{H})).$$

These design choices are not cosmetic: they enable training deep stacks and are central to why Transformers scale effectively [Vaswani et al., 2017].

From hidden states to next-token probabilities. In a decoder-only language model, the final layer outputs a sequence of hidden states $H^{(L_{\text{layers}})} \in \mathbb{R}^{L\times d}$. At a time step t (corresponding to position t in the concatenated prompt-plus-generated sequence), the model maps the hidden vector $h_t := h_t^{(L_{\text{layers}})} \in \mathbb{R}^d$ to logits over the vocabulary via an output matrix $W_U \in \mathbb{R}^{d\times|\mathcal{V}|}$ and (optionally) a bias $b_U \in \mathbb{R}^{|\mathcal{V}|}$:

$$s_t \;:=\; h_t^\top W_U + b_U \in \mathbb{R}^{|\mathcal{V}|}.$$

The component $s_{t,v}$ is the unnormalized score assigned to token $v \in \mathcal{V}$ at step t. The conditional distribution is then obtained by the softmax,

$$p_{\boldsymbol{\theta}}(y_t = v \mid y_{<t}, \mathbf{x}) \;=\; \frac{\exp(s_{t,v})}{\sum_{v'\in\mathcal{V}} \exp(s_{t,v'})}. \tag{7}$$

Often W_U is tied to the embedding table E (weight tying), which reduces parameters and can improve generalization. Under scaling and sufficient data, this parameterization supports few-shot prompting and in-context learning behaviors [Brown et al., 2020, Wei et al., 2022].

Encoder-only and encoder-decoder variants. While this course emphasizes autoregressive decoding, it is useful to contrast other Transformer families. BERT-style models are encoder-only: they process the entire input bidirectionally and are trained with masked language modeling, a denoising objective that predicts masked tokens from both left and right context [Devlin et al., 2019]. Sequence-to-sequence models adopt an encoder-decoder structure: an encoder maps an input $\mathbf{x}$ to contextual states, and a decoder generates $\mathbf{y}$ autoregressively while attending to the encoder states via cross-attention. Historically, many constrained generation problems in translation and summarization were studied in this setting because the separation between source conditioning and target decoding makes it natural to incorporate lexical and structural constraints during decoding.

4 Pretraining, scaling, and compute-data tradeoffs

Modern language models are typically trained in two stages: unsupervised pretraining on large corpora, then task adaptation (supervised fine-tuning, instruction tuning, or preference optimization). Several empirical regularities matter for constrained-aware generation.

Kaplan et al. proposed scaling laws that relate loss to model size, dataset size, and compute under regimes where optimization is stable [Kaplan et al., 2020]. Hoffmann et al. argued that, for a fixed compute budget, many models were under-trained on data, and derived compute-optimal tradeoffs that informed later LLM training recipes [Hoffmann et al., 2022]. These scaling perspectives are directly relevant to constraint-aware generation because constraint satisfaction can be viewed as requiring *effective capacity* on rare events: hard constraints often correspond to low-probability regions under the base model, so feasibility under constraints may demand either additional model capacity, additional data that covers constrained regions, or explicit inference-time constraint mechanisms.

Open technical reports such as GPT-2 [Radford et al., 2019] and LLaMA [Touvron et al., 2023] provide useful engineering context, including architectural choices, tokenization, and training setups. However, for this course the key point is conceptual: even very large base models remain stochastic predictors, so correctness and validity often require explicit constraint enforcement during decoding, reranking, or post-processing.

5 Decoding: sampling and search in autoregressive models

Given a prompt $\mathbf{x}$ and a model $p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$, generation is an iterative procedure that chooses tokens y_t sequentially. Formally, at each step t we have a distribution over next tokens as in (7), and we select a token by either (i) sampling, or (ii) approximate maximization of some global objective.

5.1 An inference view of decoding

A common objective is the maximum a posteriori (MAP) continuation

$$\hat{\mathbf{y}} \in \arg\max_{\mathbf{y}\in\mathcal{V}^T} \Big[\log p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}) + \alpha \cdot \mathrm{len}(\mathbf{y})\Big], \tag{8}$$

where the length term (or an explicit `EOS` penalty) handles variable-length decoding. Alternatively, one may aim to sample from $p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x})$ to preserve diversity. For open-ended generation, naive ancestral sampling can lead to repetitive and degenerate text [Holtzman et al., 2020], motivating a family of heuristic decoding rules that shape the next-token distribution.

5.2 Greedy decoding and temperature sampling

Greedy decoding chooses

$$y_t \in \arg\max_{v\in\mathcal{V}} \log p_{\boldsymbol{\theta}}(v \mid y_{<t}, \mathbf{x}). \tag{9}$$

This is fast but often yields generic continuations.

Temperature sampling rescales logits as $s_{t,v}/\tau$ for $\tau > 0$, defining

$$p_{\boldsymbol{\theta}}^{(\tau)}(v \mid y_{<t}, \mathbf{x}) \;:=\; \frac{\exp(s_{t,v}/\tau)}{\sum_{v'\in\mathcal{V}} \exp(s_{t,v'}/\tau)}. \tag{10}$$

As $\tau \downarrow 0$, the distribution concentrates near the greedy choice; as $\tau \uparrow \infty$, it approaches uniform over the vocabulary. Temperature is therefore a knob that trades off diversity and local likelihood.

5.3 Top-k and nucleus (top-p) sampling

Top-k sampling restricts attention to the k most likely tokens at step t:

$$\mathcal{V}_t^{(k)} := \text{top-}k \text{ tokens under } p_{\boldsymbol{\theta}}(\cdot \mid y_{<t}, \mathbf{x}), \qquad p(v) \leftarrow \frac{p(v)\,\mathbf{1}\{v \in \mathcal{V}_t^{(k)}\}}{\sum_{v'\in\mathcal{V}_t^{(k)}} p(v')}. \tag{11}$$

Algorithm 1 Nucleus (top-p) sampling

```
Input: prompt $\mathbf{x}$, model $p_{\boldsymbol{\theta}}$, threshold $p \in (0,1)$, temperature $\tau > 0$
Initialize prefix $\mathbf{y}_{<1} \leftarrow \emptyset$
for $t = 1, 2, \dots$ until EOS or length limit do
    Compute logits $s_t$ from the Transformer given $(\mathbf{x}, \mathbf{y}_{<t})$
    Form probabilities $q(v) \propto \exp(s_{t,v}/\tau)$ over $v \in \mathcal{V}$
    Sort tokens by $q(v)$ and let $\mathcal{V}_t^{(p)}$ be the smallest set with $\sum_{v \in \mathcal{V}_t^{(p)}} q(v) \geq p$
    Renormalize $\tilde{q}(v) \propto q(v)\mathbf{1}\{v \in \mathcal{V}_t^{(p)}\}$ and sample $y_t \sim \tilde{q}(\cdot)$
    Append $y_t$ to the prefix $\mathbf{y}_{<t+1} \leftarrow (\mathbf{y}_{<t}, y_t)$
end for
Return: generated sequence $\mathbf{y}$
```

Nucleus sampling (top-p) chooses the smallest set $\mathcal{V}_t^{(p)}$ whose cumulative probability mass is at least $p \in (0,1)$, then renormalizes similarly [Holtzman et al., 2020]. Unlike top-k, top-p adapts the candidate set size to the entropy of the distribution: in sharp distributions it behaves like greedy, in flat distributions it keeps more candidates.

the candidate set size to the entropy of the distribution: in sharp distributions it behaves like greedy, in flat distributions it keeps more candidates.

These heuristics can be interpreted as approximate constrained sampling with a *stepwise* feasibility set. However, because the truncation is local, it does not enforce global properties such as satisfying a grammar or including specific required substrings. This motivates explicit constrained decoding methods.

5.4 Beam search as approximate MAP

Beam search approximates (8) by maintaining a set of B partial hypotheses at each step and extending each hypothesis by likely next tokens. Let b index hypotheses and let $\ell(b)$ denote a score, often the accumulated log-probability (possibly length-normalized). Beam search proceeds by expanding each hypothesis, then keeping the B best scoring continuations.

Beam search is a generic approximate search method, not an exact inference procedure. In open-ended generation, beam search can amplify model biases toward high-probability generic text. In constrained settings, beam search is attractive because constraints can be injected by restricting allowable expansions, so the beam never enters infeasible regions.

5.5 Reranking and decoding as optimization

A practical pattern is two-stage generation: first produce a candidate set $\{\mathbf{y}^{(m)}\}_{m=1}^{M}$ using sampling or beam search, then choose the best candidate under an auxiliary scoring function:

$$\hat{\mathbf{y}} \in \arg\max_{m \in [M]} \Big[\log p_{\boldsymbol{\theta}}(\mathbf{y}^{(m)} \mid \mathbf{x}) - \lambda\, \phi(\mathbf{y}^{(m)}, \mathbf{x})\Big]. \tag{12}$$

Here ϕ may be a constraint violation score, a verifier output, a toxicity detector, or an external simulator. This is a first instance of "verifier-in-the-loop" constrained generation.

Contrastive decoding [Li et al., 2022] can be interpreted in this family: it modifies the token selection rule by preferring tokens that are simultaneously likely under an expert model and unlikely under a weaker or noisier model, biasing generation away from generic high-frequency patterns. The optimization perspective is useful for this course because it connects decoding rules to constraint energies, which later reappear in guidance, projection, proximal updates, and differentiable layers.

6 Constrained decoding via grammars and finite-state constraints

Constrained decoding enforces a global constraint set $\mathcal{C}(\mathbf{x})$ during generation, where $\mathcal{C}(\mathbf{x})$ encodes what sequences are deemed *feasible* given the prompt $\mathbf{x}$. The operational point is that decoding is incremental: at step t we have only a prefix $\mathbf{y}_{1:t}$, and we must decide which next tokens preserve the possibility of eventually reaching some $\mathbf{y} \in \mathcal{C}(\mathbf{x})$. The key technical device is therefore to represent $\mathcal{C}(\mathbf{x})$ in a way that supports fast, incremental feasibility checks on prefixes, typically via a *constraint state* that is updated alongside the language model prefix.

6.1 Prefix-closed feasibility and automata

A hard constraint set $\mathcal{C} \subseteq \mathcal{V}^T$ is useful for incremental decoding when it induces a set of feasible prefixes

$$\mathrm{Pref}(\mathcal{C}) := \{\mathbf{y}_{1:t} \;:\; \exists\, \mathbf{y}_{t+1:T} \text{ such that } (\mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) \in \mathcal{C}\}. \tag{13}$$

If a prefix $\mathbf{y}_{1:t} \notin \mathrm{Pref}(\mathcal{C})$, then no continuation can satisfy the constraint, so it is safe to prune. This observation turns constrained decoding into a standard search problem: maintain a set of partial hypotheses (prefixes), prune those that fall outside $\mathrm{Pref}(\mathcal{C})$, and score expansions using the model probabilities.

Finite-state constraints provide an especially clean representation because $\mathrm{Pref}(\mathcal{C})$ can be tracked by a finite *automaton state*. Let $A = (\mathcal{S}, \mathcal{V}, \delta, s_0, \mathcal{F})$ be a finite-state automaton (FSA), where $\mathcal{S}$ is a finite state set, s_0 is the start state, $\mathcal{F}$ is the accepting set, and $\delta : \mathcal{S} \times \mathcal{V} \to 2^{\mathcal{S}}$ is the transition relation. A sequence $\mathbf{y}$ is feasible if there exists a path from s_0 to some $s \in \mathcal{F}$ labeled by $\mathbf{y}$. Given a current automaton state s_t after reading a prefix, the set of allowed next tokens is

$$\mathcal{A}(s_t) := \{v \in \mathcal{V} : \delta(s_t, v) \neq \emptyset\}. \tag{14}$$

A constrained next-token distribution is then obtained by masking and renormalizing:

$$\pi(v \mid y_{<t}, \mathbf{x}, s_t) \;:=\; \frac{p_{\boldsymbol{\theta}}(v \mid y_{<t}, \mathbf{x})\, \mathbf{1}\{v \in \mathcal{A}(s_t)\}}{\sum_{v' \in \mathcal{A}(s_t)} p_{\boldsymbol{\theta}}(v' \mid y_{<t}, \mathbf{x})}. \tag{15}$$

This is the simplest constrained decoding rule: it guarantees that generated prefixes remain feasible, because every sampled/selected token is certified by the automaton.

Example 1 (Pure FSA feasibility: a strict output template)**.** *Consider a generator that must output a citation key in the rigid template*

$$\texttt{[A-D]}\{\texttt{0-90-9}\},$$

i.e., one capital letter in $\{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$*, then a dash, then an opening brace, then exactly two digits, then a closing brace. This is a regular language and can be represented by an FSA whose states track the current position in the template. For instance, let* s_0 *be the start state,* s_1 *the state after consuming the initial bracket* `[`*,* s_2 *after consuming the letter,* s_3 *after consuming* `-`*,* s_4 *after consuming* $\{$*,* s_5 *after the first digit,* s_6 *after the second digit, and* s_7 *the accepting state after* $\}$*. Then the allowed set* $\mathcal{A}(s_t)$ *is completely explicit:* $\mathcal{A}(s_1) = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$*,* $\mathcal{A}(s_2) = \{\texttt{-}\}$*,* $\mathcal{A}(s_4) = \{\texttt{0}, \ldots, \texttt{9}\}$*, etc. Decoding with* (15) *forces the model to emit only tokens consistent with the template; the model still controls which letter and which digits appear, but it cannot violate the structure. This is an archetypal instance of "generation as search subject to constraints": the automaton provides a prefix-feasibility oracle, and the language model provides preferences among feasible continuations.*

Algorithm 2 FSA-constrained decoding (generic skeleton)

```
1: Input: prompt $\mathbf{x}$, model $p_{\boldsymbol{\theta}}$, automaton $A = (\mathcal{S}, \mathcal{V}, \delta, s_0, \mathcal{F})$, decoding rule Select
2: Initialize prefix $\mathbf{y}_{<1} \leftarrow \emptyset$ and automaton state $s_1 \leftarrow s_0$
3: for $t = 1, 2, \dots$ until EOS or length limit do
4:     Compute next-token distribution $p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{y}_{<t}, \mathbf{x})$
5:     Compute allowed set $\mathcal{A}(s_t) = \{v : \delta(s_t, v) \neq \emptyset\}$
6:     Mask and renormalize to obtain $\pi(\cdot \mid \mathbf{y}_{<t}, \mathbf{x}, s_t)$ as in (15)
7:     Choose $y_t \leftarrow \mathsf{Select}\big(\pi(\cdot \mid \mathbf{y}_{<t}, \mathbf{x}, s_t)\big)$ (sampling, greedy, or beam expansion)
8:     Update automaton state $s_{t+1} \in \delta(s_t, y_t)$ and append token to the prefix
9: end for
10: Return: $\mathbf{y}$ (accept if final state in $\mathcal{F}$)
```

6.2 Grammar constraints

Grammars generalize FSAs. For regular constraints, an FSA is sufficient. For context-free constraints (for example matching brackets, structured queries, or typed expressions), a finite-state representation is in general impossible, but incremental feasibility can still be tracked by a *parser state*. The practical message is the same: at decoding step t, compute the set of tokens that keep the partial parse valid, mask logits accordingly, and renormalize.

To make the "state" explicit, let $\mathsf{ParseState}(\mathbf{y}_{1:t})$ denote the parser configuration after consuming the prefix, e.g., a stack of nonterminals for a predictive parser, plus any partially matched terminals. Given a grammar G, define the set of grammar-admissible next terminals at that configuration,

$$\mathcal{A}_G(\mathbf{y}_{1:t}) := \{v \in \mathcal{V} : v \text{ can be the next token in some valid derivation extending } \mathbf{y}_{1:t}\}.$$

Then constrained decoding proceeds exactly as in (15), replacing $\mathcal{A}(s_t)$ with $\mathcal{A}_G(\mathbf{y}_{1:t})$. The difference is implementation: computing $\mathcal{A}_G$ typically requires parser actions, and the "mask" is grammar-induced rather than automaton-induced.

In token-based LLMs, grammar constraints require care because tokenization breaks strings into subwords. A grammar written at the character level must be aligned with the token vocabulary, often by compiling the grammar into a token-level automaton or by running a character-level parser while checking whether candidate tokens can be extended to a valid character string. This compilation step is itself a constraint engineering decision, and it is one reason why constrained generation often prefers domain-specific tokenizations (for example code tokens or JSON tokens) when strict syntactic validity is required.

Example 2 (Context-free feasibility: balanced parentheses and well-formed expressions)**.** *Consider generating arithmetic expressions over digits and operators with the additional requirement that parentheses are balanced and properly nested, e.g.,*

$$\texttt{(3+5)*(7-2)} \quad \textit{is valid, while} \quad \texttt{(3+5))*7} \quad \textit{is invalid.}$$

Balanced parentheses are a canonical context-free constraint: validity cannot be captured by any finite-state automaton because it requires unbounded counting and nesting. A simple grammar is

$$E \to E \texttt{ + } T \mid E \texttt{ - } T \mid T, \qquad T \to T \texttt{ * } F \mid T \texttt{ / } F \mid F, \qquad F \to \texttt{(}E\texttt{)} \mid \texttt{num}.$$

An incremental parser maintains a state that includes, at minimum, the stack of expected closing parentheses and the current nonterminal expansion context. After producing the prefix `(3+5)*`*, the parser state implies that the next token cannot be* `)` *(there is no open parenthesis to close at that point) and cannot be an operator (we just emitted* `*`*), but it can be* `(` *or a digit starting a* `num`*. Thus the grammar-induced admissible set $\mathcal{A}_G(\mathbf{y}_{1:t})$ rules out syntactically impossible* *continuations before the language model scores them. In constrained decoding, the language model decides* which *feasible continuation is most plausible (e.g., (versus 7), while the grammar enforces global well-formedness by construction.*

6.3 Lexically constrained decoding

Lexical constraints require that a set of words or phrases appear in the generated sequence. This is common in translation (must include named entities), summarization (must include key facts), and domain generation (must include required fields). Lexical constraints can be represented by an automaton that tracks which constraints have been satisfied so far. If there are C required items, a conceptually simple representation is a state variable $c \in \{0,1\}^C$ that encodes which items have appeared.

The central difficulty is not feasibility per se but search complexity: once constraints are introduced, naive beam search can easily "use up" the beam on fluent but constraint-violating hypotheses, and may fail to ever place any probability mass on prefixes that complete all required items. The role of lexically constrained methods is therefore to force the search to *reserve capacity* for constraint-progressing hypotheses.

Grid Beam Search (GBS). Grid Beam Search (GBS) [Hokamp and Liu, 2017] constructs a grid indexed by time t and the number k of constraints satisfied so far. It maintains a separate beam $B_{t,k}$ for each cell, where $B_{t,k}$ contains the best-scoring prefixes of length t that have satisfied exactly k constraints. Expansions are only allowed if they correctly update the constraint-count index. Intuitively, the grid prevents the search from collapsing onto high-probability prefixes that make no progress on constraints, because there is always an explicitly maintained beam for larger k.

Dynamic Beam Allocation (DBA). Dynamic Beam Allocation (DBA) [Post and Vilar, 2018] improves efficiency by using a single global beam budget B and allocating it adaptively across constraint states, rather than populating all grid cells with the same fixed width. In practice, DBA maintains hypotheses grouped by their constraint satisfaction state and reallocates beam slots toward groups that still need capacity to complete remaining constraints. The key effect is that DBA achieves most of the robustness of GBS at significantly lower computational cost, especially when C is large or when the constraint automaton has many states.

Example 3 (Lexically constrained decoding in translation: forcing named entities)**.** *Consider translating the source sentence* $\mathbf{x} =$ `‘‘I met Dr. Chen in Charlottesville.’’` *into Italian, with the hard lexical constraint that the output must contain the name* `‘‘Chen’’` *(to avoid mistranslating or dropping the named entity). Let the required phrase set be* $\{$`Chen`$\}$*, so* $C = 1$ *and the constraint state is* $c \in \{0, 1\}$*, where* $c = 1$ *indicates that the phrase has appeared.*

In unconstrained beam search, many high-probability prefixes may commit early to a paraphrase that omits the name, and once the search has pruned all hypotheses that still allow inserting `Chen` *naturally, the constraint becomes impossible to satisfy. GBS prevents this by maintaining two beams at each time:* $B_{t,0}$ *for prefixes that have not yet produced* `Chen` *and* $B_{t,1}$ *for prefixes that have. Even if* $B_{t,0}$ *contains the most fluent prefixes, the algorithm keeps a dedicated beam for* $c = 1$ *so that once a hypothesis emits* `Chen` *it is not immediately drowned out. DBA achieves a similar effect while letting the width devoted to* $c = 0$ *versus* $c = 1$ *shift over time, for example allocating more width to* $c = 0$ *early (exploration) and reserving enough width to* $c = 1$ *as the length budget shrinks (completion pressure).*

This example highlights a recurring theme in constrained-aware generation: enforcing a global requirement is often less about masking individual tokens and more about managing search resources so that constraint-satisfying hypotheses remain alive until they can be completed.

Example 4 (Lexical constraints as the first "policy" constraint)**.** *Consider an email assistant that must generate a meeting confirmation containing two mandatory phrases: (i) `''Rice Hall 340''` and (ii) `''Thursday at 9:30AM''`. Let $\mathcal{C}(\mathbf{x})$ be the set of continuations that contain both phrases at least once. A constrained decoder can maintain a state variable $c \in \{0,1\}^2$ and update it when a phrase is completed. Beam search can then be run over pairs $(\textit{prefix}, c)$, pruning any expansion that makes it impossible to satisfy the remaining phrases within the length budget. This is a minimal instance of treating generation as search under a symbolic policy constraint.*

6.4 Complexity and failure modes

Constrained decoding trades probability mass for validity. If $\mathcal{C}(\mathbf{x})$ is too strict relative to the base model, the feasible set may have extremely small probability, leading to brittle search and low-quality outputs. Algorithmically, constraints increase decoding complexity because the search state must track both the language model prefix and the constraint state (for example an automaton state or a constraint satisfaction vector). For beam-based methods, complexity typically scales as $O(BT)$ in the unconstrained case, and as $O(BT \cdot |\mathcal{S}|)$ or $O(BT \cdot 2^C)$ depending on the constraint representation.

From the course perspective, this motivates three later themes. First, we want constraints to be represented in a way that admits efficient incremental feasibility checks. Second, we want the base model to assign nontrivial mass to the feasible set, which connects to training-time constraint injection. Third, we want principled ways to balance soft and hard constraints, which connects to optimization tools such as penalties, Lagrangians, and splitting methods.

7 Summary and outlook

Autoregressive Transformers define a tractable likelihood via the factorization $p_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}) = \prod_t p_{\boldsymbol{\theta}}(y_t \mid y_{<t}, \mathbf{x})$ and implement it with a causal self-attention architecture. Decoding is then an inference procedure, either sampling or approximate optimization, and the decoding step is where constraints can be enforced directly as feasibility masks or as reranking energies.

The constrained decoding mechanisms in this lecture are intentionally simple: grammars, finite-state constraints, and lexical constraints. Their value is that they provide a clean, operational example of "generation as search subject to constraints" before we introduce the optimization toolbox. In later lectures, we will revisit the same pattern in continuous domains, where constraint enforcement often proceeds through projection, proximal operators, or guidance forces.

References

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. In *ICLR*, 2020.

S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *NeurIPS*, 2015.

M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. In *ICLR*, 2016.

L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *arXiv:1511.01844*, 2016.

C. Hokamp and Q. Liu. Lexically constrained decoding for sequence generation using grid beam search. In *ACL*, 2017.

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

Y. Li, Y. Liang, K. Zhang, and M. Ranzato. Contrastive decoding: Open-ended text generation as optimization. In *ACL*, 2022.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv:2001.08361*, 2020.

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. Henderson, A. Menick, et al. Training compute-optimal large language models. *arXiv:2203.15556*, 2022.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, et al. LLaMA: Open and efficient foundation language models. *arXiv:2302.13971*, 2023.

J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, E. H. Chi, T. Hashimoto, P. Liang, and J. Dean. Emergent abilities of large language models. *arXiv:2206.07682*, 2022.