CTRL-CFG: TRACTABLE CONSTRAINED DECODING WITH LL(1) GRAMMARS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language models often need to generate structured outputs that satisfy a context-free grammar such as JSON or Python. Steering a language model towards satisfying such a grammar requires marginalizing the probability that a future continuation will be grammatical. Hidden Markov models have proven effective as tractable surrogates for this marginalization, but existing methods focused on regular languages. We extend this line of work to LL(1) grammars, a widely used subset of context-free grammars. Unfortunately, even with a tractable surrogate, constrained generation with unambiguous grammars is still cubic time in the sequence length. We hence develop **CTRL-CFG**, a linear-time approximation that marginalizes only over a fixed look-ahead window. Experiments on program synthesis and molecule generation illustrate the potential of this approach as a step towards more broadly usable tractable constrained decoding.

1 INTRODUCTION

Large language models (LLMs) are widely used for structured generation tasks such as code generation, formal theorem proving, and tool calling with JSON APIs. In all of these settings, the LLM-generated text must conform to a *context-free grammar* (CFG). Unfortunately, LLMs are known to produce syntactically invalid outputs in practice (Lu et al., 2023; Geng et al., 2025; Zuo et al., 2025). Theoretical results have moreover indicated that this is an intrinsic expressivity limitation of bounded-depth transformer architectures (Merrill, 2025, Sec. 3.4.2).

Constrained decoding makes it impossible for outputs to violate a constraint $\mathcal{C}$, by conditioning the next-token distribution $p(x_t \mid x_{0:t})$ on said constraint. Using Bayes' rule, we have

$$p(x_t \mid x_{0:t}, \mathcal{C}) \propto p(x_t \mid x_{0:t}) \cdot p(\mathcal{C} \mid x_{0:t+1}). \tag{1}$$

The last term $p(\mathcal{C} \mid x_{0:t+1})$ is the probability that the prefix $x_{0:t+1}$ will be completed into a valid string x. This *membership probability* marginalizes over all possible continuations $x_{t+1:n}$, and is hence intractable on a transformer. Practical implementations typically resort to *locally constrained decoding* (LCD), by masking the next-token probabilities (Scholak et al., 2021; Beurer-Kellner et al., 2024). This means the membership probability is replaced with whether $x_{0:t+1}$ can still be extended to a grammatical string: $p(x \in \mathsf{L} \mid x_{0:t+1}) \approx [\![\exists y : x_{0:t+1}y \in \mathsf{L}]\!]$. Although efficient, LCD is a heavily biased approximation: the existence of a continuation says nothing about its probability.

At the other extreme, *sampling-based estimators* based on rejection sampling (Park et al., 2024; Melcer et al., 2026), sequential Monte Carlo (Lew et al., 2023; Loula et al., 2025; Lipkin et al., 2025), and MCMC (Gonzalez et al., 2025) approach the true posterior of Equation 1. However, these methods are expensive as they require multiple forward passes through the language model.

Zhang et al. (2024) proposed a middle ground for constrained decoding: *compute the membership term of Equation 1 on a tractable surrogate*. Specifically, they showed that on a hidden Markov model (HMM), computing the membership probability takes linear time in the sequence length. This keeps the computational overhead limited compared to raw LLM inference, while considerably improving the approximation. The method of Zhang et al. (2024) is restricted to regular languages, however, while many practical languages are inherently context-free. For this reason, **our work extends the surrogate-model approach to context-free grammars.**

The greater expressivity of CFGs makes tractable marginalization considerably more difficult than on regular languages. As we discuss in Section 3, constrained decoding with a tractable surrogate

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

still takes cubic time for unambiguous CFGs (Marzouk & de La Higuera, 2022; Zhang et al., 2025). This makes the tractable surrogate approach of Zhang et al. (2024) impractical for CFGs (Scribner et al., 2026). We hence propose CTRL-CFG, a tractable constrained decoding method using a *look-ahead window* that computes the membership probability over only the next k tokens (Section 4). This look-ahead window gives an upper bound on the membership probability that lies between exact marginalization and locally constrained decoding. We prove that for LL(1) grammars, CTRL-CFG allows for linear-time constrained decoding. Experiments on program synthesis and molecule generation indicate the practical potential of CTRL-CFG, showing how bounded look-ahead trades off sample quality against decoding cost (Section 6).

2 PRELIMINARIES

Notation. We fix a finite set of tokens Σ and write $x_{0:n} = x_0 \dots x_{n-1}$ for a sequence of n tokens, such that $x_{i:i} = \epsilon$ and $x_{i:i+1} = x_i$. The concatenation of two sequences x and y is denoted xy. We write Σ^n for the set of all length-n sequences over Σ and Σ^* for all finite sequences over Σ, including the empty sequence ϵ. A *language* L is a set of sequences. The *Brzozowski derivative* $x^{-1}\mathsf{L} = \{y \in \Sigma^* \mid xy \in \mathsf{L}\}$ is the language of all possible continuations of a prefix $x \in \Sigma^*$.

Definition 1 (Context-free grammar)**.** *A context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ consists of a set of nonterminals $\mathcal{N}$, a set of terminals Σ, a set of production rules $\mathcal{R} \subseteq \mathcal{N} \times (\Sigma \cup \mathcal{N})^*$, and a start symbol $S \in \mathcal{N}$.*

The rewrite relation $\Rightarrow_G$ of G applies a production $A \to \beta$ to a nonterminal A in a sequence.

$$\{\alpha A \gamma \Rightarrow_G \alpha\beta\gamma \mid (A \to \beta) \in \mathcal{R},\ \alpha, \gamma \in (\Sigma \cup \mathcal{N})^*\}$$

The language of G is $\mathsf{L}(G) \coloneqq \{x \in \Sigma^* \mid S \overset{*}{\Rightarrow}_G x\}$, where $\overset{*}{\Rightarrow}_G$ is the reflexive-transitive closure of $\Rightarrow_G$. To simplify later algorithms, we use a *binary normal form* (BNF). As explained in Appendix A, any grammar may be converted into BNF with only linear overhead.

Definition 2 (Binary normal form)**.** *A grammar $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ is in binary normal form if*

- *rules in $\mathcal{R}$ are of the form $A \to \epsilon$, $A \to a$, $A \to B$, or $A \to BC$, with $a \in \Sigma$ and $A, B, C \in \mathcal{N}$,*
- *every nonterminal $A \in \mathcal{N}$ is reachable from S and derives a terminal sequence.*

For generation, we assume each sequence $x \in \mathsf{L}(G)$ contains the special end-of-sequence eos as its final token. This makes the language *prefix-free*, meaning no sequence is a proper prefix of another.

A grammar G is *unambiguous* when each sequence $x \in \mathsf{L}(G)$ has a unique leftmost derivation. A grammar is furthermore in LL(1) when the current nonterminal and the next token determine which production to apply. This is formalized with the first and follow sets.

Definition 3 (LL(1))**.** *For each $\alpha \in (\Sigma \cup \mathcal{N})^*$, the set $\mathsf{first}(\alpha) \subseteq \Sigma \cup \{\epsilon\}$ contains the terminals that can begin a derivation of α, together with ϵ if α is nullable.*

$$\mathsf{first}(\alpha) = \{a \in \Sigma \mid \alpha \overset{*}{\Rightarrow}_G a\beta\} \cup \{\epsilon \mid \alpha \overset{*}{\Rightarrow}_G \epsilon\} \quad \textit{with } \beta \in (\Sigma \cup \mathcal{N})^*$$

For each $A \in \mathcal{N}$, the set $\mathsf{follow}(A) \subseteq \Sigma$ contains the terminals that can immediately follow A.

$$\mathsf{follow}(A) = \{a \in \Sigma \mid S \overset{*}{\Rightarrow}_G \beta_1 A a \beta_2\} \quad \textit{with } \beta_1, \beta_2 \in (\Sigma \cup \mathcal{N})^*$$

A grammar is LL(1) *when every pair of distinct productions $A \to \alpha$ and $A \to \beta$ satisfies both (1) $\mathsf{first}(\alpha) \cap \mathsf{first}(\beta) = \varnothing$, and (2) if $\epsilon \in \mathsf{first}(\alpha)$ then $\mathsf{first}(\beta) \cap \mathsf{follow}(A) = \varnothing$.*

LL(1) grammars are a proper subset of unambiguous grammars, which are themselves a proper subset of CFGs. Nonetheless, they are a popular class of grammars as they admit linear-time parsing.

Hidden Markov models. An HMM defines a joint distribution over a sequence $x_{0:n}$ and latent states $z_{0:n}$, where the latent states form a first-order Markov chain. So z_t conditioned on z_{t-1} is independent of earlier latent states. Moreover, an observation x_t conditioned on z_t is independent of all other latent and observed variables. The transition and emission probabilities are time-invariant.

$$p(x_{0:n}, z_{0:n}) = p(z_0)\, p(x_0 \mid z_0) \prod_{i=1}^{n-1} p(z_i \mid z_{i-1})\, p(x_i \mid z_i).$$

These conditional independence assumptions let us iteratively compute the marginal $p(x_{0:n})$ in $O(nh^2)$ via the forward algorithm (Rabiner & Juang, 1986), where h is the number of states for each latent variable z_i. Starting from $p(x_{0:1}, z_0) = p(z_0)p(x_0 \mid z_0)$, we compute $p(x_{0:t+1}, z_t)$ using $p(x_{0:t}, z_{t-1})$ as

$$p(x_{0:t+1}, z_t) = p(x_t \mid z_t)\, p(x_{0:t}, z_t) = p(x_t \mid z_t) \sum_{z_{t-1}=1}^{h} p(z_t \mid z_{t-1})\, p(x_{0:t}, z_{t-1}).$$

3 Exact CFG Marginalization

As shown in Equation 1, to sample a sequence x conditioned on the grammar G, we need to compute the probability that x is a member of the language $\mathsf{L}(G)$. Fixing a sequence length of $n \geq 1$ tokens[1], we can write the membership probability of a grammar G as the expectation

$$p_{\mathsf{LLM}}(x_{t:n} \in x_{0:t}^{-1}\mathsf{L}(G) \mid x_{0:t}) = \mathbb{E}_{x_{t:n} \sim p_{\mathsf{LLM}}(\cdot \mid x_{0:t})}[\![S \overset{*}{\Rightarrow}_G x_{0:n}]\!]. \tag{2}$$

For now, we simplify this membership probability to $p_{\mathsf{LLM}}(x_{0:n} \in \mathsf{L}(G))$ by ignoring the prefix (i.e., $t = 0$). Because Equation 2 is intractable, Zhang et al. (2024) proposed to substitute the LLM distribution p_{LLM} with an HMM distribution p_{HMM}. For regular languages, the marginal can then be computed in linear time. On CFGs it unfortunately remains hard in general (Bertoni et al., 1991).

Theorem 1 (Marzouk & de La Higuera 2022, p. 2)**.** *Given a context-free grammar G, an HMM, and a sequence length n, computing $p_{\mathsf{HMM}}(x_{0:n} \in \mathsf{L}(G))$ is #P-hard.*

On the more restricted class of unambiguous grammars, however, marginalization is polytime.

Theorem 2 (Marzouk & de La Higuera 2022, Sec. 2)**.** *Given an unambiguous context-free grammar $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ and an HMM with h states, computing $p_{\mathsf{HMM}}(x_{0:n} \in \mathsf{L}(G))$ takes $O(n^2h^3|\mathcal{R}|)$ time and $O(nh^2|\mathcal{N}|)$ space.*

Marzouk & de La Higuera (2022) achieve this complexity with a CYK-style chart recursion. The algorithm centers around the probability $p(A \overset{*}{\Rightarrow}_G x_{0:\ell},\, z_\ell \mid z_0)$ that the span $x_{0:\ell}$ is derivable from A while the HMM latent walks from z_0 to z_ℓ. Due to the BNF assumption, this probability equals the aggregation over terminal, unary, and binary rules expanding A and the position $0 \leq \ell' \leq \ell$ where binary rules are split. Unambiguity of the grammar guarantees that at most one production and split derives a given $x_{0:\ell}$, meaning we can sum the mutually exclusive probabilities.

$$\begin{aligned} p(A \overset{*}{\Rightarrow} x_{0:\ell},\, z_\ell \mid z_0) = [\![\ell = 1]\!] \cdot \sum_{(A \to a) \in \mathcal{R}} p(a \mid z_0)\, p(z_1 \mid z_0) + \sum_{(A \to B) \in \mathcal{R}} p(B \overset{*}{\Rightarrow} x_{0:\ell},\, z_\ell \mid z_0) \\ + \sum_{(A \to BC) \in \mathcal{R}} \sum_{\ell'=0}^{\ell} \sum_{z_{\ell'}=1}^{h} p(B \overset{*}{\Rightarrow} x_{0:\ell'}, z_{\ell'} \mid z_0) \cdot p(C \overset{*}{\Rightarrow} x_{\ell':\ell}, z_\ell \mid z_{\ell'}). \end{aligned}$$

In the base case where $\ell = 0$, $p(A \overset{*}{\Rightarrow}_G x_{0:0}, z_0 \mid z_0) = [\![A \overset{*}{\Rightarrow}_G \epsilon]\!]$. Algorithm 2 in Appendix B computes the above probability for each nonterminal $A \in \mathcal{N}$ and each length $0 \leq \ell \leq n$. Unary rules and binary rules with a nullable child introduce dependencies at the same length, which we evaluate in topological order.[2] As the HMM is homogeneous, only the length of a marginalized span matters and not the absolute position, so we can use $C \overset{*}{\Rightarrow} x_{0:\ell-\ell'}$ instead of $C \overset{*}{\Rightarrow} x_{\ell':\ell}$. Finally, we obtain the desired membership probability by marginalising away the boundary HMM states:

$$p_{\mathsf{HMM}}(x_{0:n} \in \mathsf{L}(G)) = \sum_{z_0=1}^{h} p(z_0) \sum_{z_n=1}^{h} p(S \overset{*}{\Rightarrow} x_{0:n}, z_n \mid z_0).$$

From the tractable circuits perspective (Choi et al., 2020), this algorithm multiplies the Boolean circuit of the CFG with the HMM probabilistic circuit before marginalizing. For inhomogeneous HMMs without position invariance, the complexity becomes cubic in the sequence length, recovering the circuit construction of Zhang et al. (2025, Thm. 4.2).

[1] In practice, we rarely know the exact length n of the final sequence. However, the above algorithm extends straightforwardly to sequences of *at most* n tokens, for example by adding padding tokens.

[2] A cycle would give a nontrivial derivation $A \overset{*}{\Rightarrow} A$, contradicting unambiguity. See Appendix B.1.

4 BOUNDED LOOK-AHEAD MARGINALIZATION

For constrained decoding, we need the membership probability *each time we sample a token*. So using the quadratic marginalization of Theorem 2 at each token results in an $O(n^3)$ time complexity to sample a length-n sequence, which is prohibitive for long sequences. We hence propose an approximation to achieve linear-time inference, by *only looking ahead a fixed number of tokens* past the current prefix. We formalize this approximation using the *prefix language* of a grammar.

$$\mathsf{PL}(G) = \{x \mid \exists y \in \Sigma^* : xy \in \mathsf{L}(G)\}$$

In other words, $\mathsf{PL}(G)$ contains all sequences that are a prefix of a sequence in $\mathsf{L}(G)$. Marginalizing over the length-k window $x_{t:t+k}$ beyond the prefix $x_{0:t}$ then gives the following upper bound.

$$\begin{aligned}
p(x_{t:n} \in x_{0:t}^{-1}\mathsf{L}(G) \mid x_{0:t}) &= \sum_{x_{t:t+k}} p(x_{t:t+k} \mid x_{0:t}) \sum_{x_{t+k:n}} p(x_{t+k:n} \mid x_{0:t+k}) [\![x_{0:n} \in \mathsf{L}(G)]\!] \\
&\leq \sum_{x_{t:t+k}} p(x_{t:t+k} \mid x_{0:t}) \sum_{x_{t+k:n}} p(x_{t+k:n} \mid x_{0:t+k}) [\![x_{0:t+k} \in \mathsf{PL}(G)]\!] \\
&= \sum_{x_{t:t+k}} p(x_{t:t+k} \mid x_{0:t}) [\![x_{0:t+k} \in \mathsf{PL}(G)]\!] \\
&= p(x_{t:t+k} \in x_{0:t}^{-1}\mathsf{PL}(G) \mid x_{0:t}) \qquad (3)
\end{aligned}$$

In the second step, we use that $[\![x_{0:n} \in \mathsf{L}(G)]\!] \leq [\![x_{0:t+k} \in \mathsf{PL}(G)]\!]$. This upper bound improves monotonically as k increases, and can be replaced with the exact membership probability when $k \geq n - t$. At the other extreme, $k = 0$ recovers locally constrained decoding. But to use this window upper bound, we need to marginalize over the prefix language $\mathsf{PL}(G)$ instead of over $\mathsf{L}(G)$. This poses a challenge, as unambiguity does not guarantee that each prefix has a unique parse.

Theorem 3 (Hardness of prefix marginalization)**.** *Given an unambiguous grammar G, a window length k, and an HMM, computing $p_{\mathsf{HMM}}(x_{0:k} \in \mathsf{PL}(G))$ is #P-hard.*

To recover tractability, we turn to LL(1) grammars. Intuitively, LL(1) provides a "prefix-unambiguity", as it can decide how to parse using only a single token look-ahead. In the next subsection, we explain how LL(1) in combination with the upper bound leads to linear-time constrained sampling.

4.1 CTRL-CFG: LINEAR-TIME CONSTRAINED GENERATION

To turn the marginalization over a prefix language into one over an ordinary language, we use prefix grammars. The prefix grammar $\mathrm{Pre}(G)$ is a grammar for the prefix language of G, i.e., $\mathsf{PL}(G) = \mathsf{L}(\mathrm{Pre}(G))$. We roughly follow Pasti et al. (2026, Def. 1).

Definition 4 (Prefix grammar)**.** *Given a grammar $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ in BNF, define $\mathrm{Pre}(G) = (\mathcal{N} \cup \mathcal{N}' \cup \{\hat{S}\}, \Sigma, \mathcal{R} \cup \mathcal{R}', \hat{S})$ with $\mathcal{N}' = \{A' \mid A \in \mathcal{N}\}$ the new nonterminals and $\hat{S}$ a new start symbol. $\mathcal{R}'$ contains (1) $A' \to a$ for each terminal rule $A \to a$, (2) $A' \to B'$ for each unary rule $A \to B$, (3) $A' \to B'$ and $A' \to BC'$ for each binary rule $A \to BC$, (4) $\hat{S} \to \epsilon$, and (5) $\hat{S} \to S'$.*

As implied by Theorem 3, the unambiguity of G does not imply that $\mathrm{Pre}(G)$ is also unambiguous. For LL(1) grammars, however, the transformation preserves unambiguity.

Theorem 4. *If G is an LL(1) grammar, its prefix grammar $\mathrm{Pre}(G)$ is unambiguous.*

So on LL(1) grammars, we can compute $p(x \in \mathsf{L}(\mathrm{Pre}(G)))$ instead of $p(x \in \mathsf{PL}(G))$ using the method of Section 3. There remain two gaps to use the upper bound of Equation 3 for constrained sampling: (1) we need to condition the membership probability on the prefix $x_{0:t}$, and (2) we need this conditioned membership probability for each possible next token.

Conditioning. Because G is in LL(1), there is a unique leftmost rewrite $S \overset{*}{\Rightarrow}_G x_{0:t}\, \beta \overset{*}{\Rightarrow}_G x_{0:n}$ for each prefix $x_{0:t}$. Here, $\beta \in \mathcal{N}^*$ is the parser stack after parsing the prefix $x_{0:t}$. It follows that

$$p(x_{t:n} \in x_{0:t}^{-1}\mathsf{L}(G) \mid x_{0:t}) = \mathbb{E}_{x_{t:n} \sim p(\cdot \mid x_{0:t})} [\![S \overset{*}{\Rightarrow}_G x_{0:n}]\!] = \mathbb{E}_{x_{t:n} \sim p(\cdot \mid x_{0:t})} [\![\beta \overset{*}{\Rightarrow}_G x_{t:n}]\!].$$

Algorithm 1 CTRL-CFG inference

Require: LL(1) grammar G and window size $k \geq 1$.
Require: Language model p_{LLM} and HMM p_{HMM} over the same vocabulary as G.
Initialization.
Convert the grammar G to BNF. (Appendix A)
Construct the prefix grammar $\mathrm{Pre}(G)$. (Definition 4)
Cache the HMM matrices for each $A \in \mathcal{N}$ and the prefix vectors for A'. (Appendix B.2)
Set $t \leftarrow 0$, parser stack $\beta \leftarrow S$, and HMM state $q_0(z) \leftarrow p_{\mathsf{HMM}}(z_0 = z)$.
Generation.
while $t = 0$ **or** $x_{t-1} \neq \mathsf{eos}$ **do**
  Read $p_{\mathsf{HMM}}(\mathcal{C} \mid x_{0:t})$ from the current stack cache. (Equation 3)
  Run the backward pass to get $p_{\mathsf{HMM}}(\mathcal{C} \mid x_{0:t}a)$ for each $a \in \Sigma$. (Equation 6)
  Sample $x_t \sim p_{\mathsf{LLM}}(a \mid x_{0:t})\, p_{\mathsf{HMM}}(\mathcal{C} \mid x_{0:t}a)$. (Equation 1)
  Advance the LL(1) parser of G through x_t, updating suffix caches on pushes and pops.
  $t \leftarrow t + 1$
  $q_t(z) \leftarrow p_{\mathsf{HMM}}(z_t = z \mid x_{0:t})$
return the sampled sequence $x_{0:t}$.

In other words, the parser stack β encodes the grammatical constraints on continuations of the prefix $x_{0:t}$. We construct a new grammar G_β, which is identical to G except that the start symbol is S_β with an additional rule $S_\beta \to \beta$. The prefix grammar of G_β has start symbol $\hat{S}_\beta$ and language $\mathsf{L}(\mathrm{Pre}(G_\beta)) = x_{0:t}^{-1}\mathsf{PL}(G)$. By including the HMM latent states as in Section 3, we get

$$p_{\mathsf{HMM}}(x_{t:t+k} \in x_{0:t}^{-1}\mathsf{PL}(G), z_{t+k} \mid z_t, x_{0:t}) = p_{\mathsf{HMM}}(\hat{S}_\beta \overset{*}{\Rightarrow}_{\mathrm{Pre}(G_\beta)} x_{t:t+k}, z_{t+k} \mid z_t). \tag{4}$$

Caching. The above probability does not need to be computed from scratch at each token. All the probabilities $p_{\mathsf{HMM}}(A \overset{*}{\Rightarrow}_{\mathrm{Pre}(G)} x_{0:\ell}, z_\ell \mid z_0)$ for $0 \leq \ell \leq k$ and each nonterminal A of $\mathrm{Pre}(G)$ can be precomputed and cached. Only the additional rule $S_\beta \to \beta$ changes as the parser stack is modified. So each time a nonterminal A is pushed on the parser stack β, we can compute the new membership probability, for $1 \leq \ell \leq k$, as

$$\begin{aligned} p_{\mathsf{HMM}}(\hat{S}_{A\beta} \overset{*}{\Rightarrow}_{\mathrm{Pre}(G_{A\beta})} x_{0:\ell} \mid z_0) &= p_{\mathsf{HMM}}(A' \overset{*}{\Rightarrow}_{\mathrm{Pre}(G)} x_{0:\ell} \mid z_0) \\ &\quad + \sum_{j=0}^{\ell-1} \sum_{z_j=1}^{h} p_{\mathsf{HMM}}(A \overset{*}{\Rightarrow}_G x_{0:j}, z_j \mid z_0)\, p_{\mathsf{HMM}}(\hat{S}_\beta \overset{*}{\Rightarrow}_{\mathrm{Pre}(G_\beta)} x_{j:\ell} \mid z_j). \end{aligned} \tag{5}$$

As the parser only pushes nonterminals to the top of the stack, the final HMM state can be summed out for primed nonterminals. So only a vector over internal states needs to be cached for each length instead of the full matrix.

All-Marginals. When sampling the next token, we need the membership probability conditioned on $x_{0:t}a$ for each single token $a \in \Sigma$. A naive solution would be problematic for large vocabularies. Fortunately, dynamic programming can reuse computation to obtain all the marginals in a single forward and backward pass. Computing all marginals with dynamic programming is essentially the backpropagation algorithm (Darwiche, 2003; Maene & De Raedt, 2025), while the naive solution corresponds to taking the forward gradient. After computing the membership probability in a forward pass, a backward pass computes

$$p_{\mathsf{HMM}}(x_{t+1:t+k} \in (x_{0:t}a)^{-1}\mathsf{PL}(G) \mid x_{0:t}a) = \frac{\partial p_{\mathsf{HMM}}(x_{t:t+k} \in x_{0:t}^{-1}\mathsf{PL}(G) \mid x_{0:t})}{\partial p_{\mathsf{HMM}}(x_t = a \mid x_{0:t})} \tag{6}$$

Here the derivative varies the first-token mixture weights while holding the conditional continuation distributions fixed (see Appendix B.3). The backward pass reuses the precomputed nonterminal probabilities and cached stack-suffix probabilities from Equation 5, sharing these computations across all candidate tokens.

Complexity. The stack push of Equation 5 computes k cached vectors, each summing over at most k split points with an $h \times h$ matrix–vector product, and therefore costs $O(k^2h^2)$ time and $O(kh)$

space. A stack pop only needs to restore the cached suffix, and is hence $O(1)$. As explained in Appendix C.3, there are at most $O(n|\mathcal{N}|)$ stack pushes when parsing a length-n sequence. So we get $O(nk^2h^2|\mathcal{N}|)$ time in total for the stack updates. The backward pass likewise takes $O(k^2h^2|\mathcal{R}|)$ time per token, in addition to the one-time $O(k^2h^3|\mathcal{R}|)$ precomputation time. The window cache uses $O(kh^2|\mathcal{N}|)$ space, while the parser stack uses at most $O(nkh|\mathcal{N}|)$ space. Together, these give the following bound for Algorithm 1. A proof is given in Appendix C.3.

Theorem 5. *Given an LL(1) grammar $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$, an HMM with h latent states, and a window size k, CTRL-CFG samples a length-n sequence in $O(nk^2h^2|\mathcal{R}| + k^2h^3|\mathcal{R}|)$ time and $O(nkh|\mathcal{N}| + kh^2|\mathcal{N}|)$ space.*

5 Related Work

Grammar-constrained decoding. Per-token grammar masking (Scholak et al., 2021; Beurer-Kellner et al., 2024; Park et al., 2025; Dong et al., 2025) is the workhorse of grammar-constrained decoding and a special case of our method at k=0. Beam search and lexically constrained decoding (Hokamp & Liu, 2017) search for high-probability sequences that are part of the grammar. Sampling-based methods constrain the LLM distribution using rejection sampling (Park et al., 2024), sequential Monte Carlo (Lipkin et al., 2025), or MCMC (Miao et al., 2019; Gonzalez et al., 2025). Qin et al. (2022) apply Langevin dynamics, and Ahmed et al. (2026) compile the constraint into a circuit and resample locally. Mündler et al. (2025) adapt rejection sampling to diffusion language models. All of these methods either pay a many-sample cost per step or accept a token-level approximation. Our linear-time look-ahead window sits between these regimes.

HMM-based surrogates. Zhang et al. (2023) and Zhang et al. (2024) proposed the use of HMMs as a tractable surrogate for constrained LM decoding, with constraints in conjunctive normal form and as regular expressions, respectively. Closely related, Marzouk & de La Higuera (2022) give an exact quadratic-time algorithm for CFG marginals over HMMs (see Section 3). We build on this for our linear-time look-ahead upper bound. Zhang et al. (2025) give a more general cubic circuit construction for CFGs. Baiget et al. (2026) apply this to learn distributions over grammars for fuzzing. The complexity of probabilistic word problems for context-free grammars is also analyzed by Amarilli et al. (2025). Our contribution is to turn this line of work into a linear-time constrained decoding method for LL(1) grammars.

Concurrent to our work, Scribner et al. (2026) also investigated constrained decoding on CFGs with HMMs, by proposing a marginalization procedure for LR(k) grammars. However, the existing marginalization method for unambiguous grammars (cf. Section 3) is more general. Moreover, it is linear in the grammar size, while the approach of Scribner et al. (2026) is quadratic in the number of automaton states. Dang et al. (2026) use the approach of Zhang et al. (2024) as a proposal for sequential Monte Carlo sampling. A notable advantage is that they are no longer restricted to unambiguous grammars. This same idea could also be used to extend our method to grammars beyond LL(1).

6 Experiments

We evaluate CTRL-CFG on three structured decoding tasks: molecule generation, regular-expression synthesis, and text-to-SQL generation. Our evaluation investigates whether bounded look-ahead makes surrogate-based constrained decoding applicable beyond regular languages, and how its benefits and computational costs depend on the task, window size, and surrogate model. Appendix D includes additional details on training and hyperparameters. All experiments are conducted on an NVIDIA L40S with 48GB VRAM.

6.1 Baselines

We compare against *unconstrained sampling* from the base model and *locally constrained decoding* (LCD), which greedily masks tokens. We implement LCD with XGrammar (Dong et al., 2025), and use Gemma 4 E2B-it (Google, 2026) and Llama 3.1 8B Instruct (Grattafiori et al., 2024) as representative open-weight models. We further include *rejection sampling*, which repeatedly samples

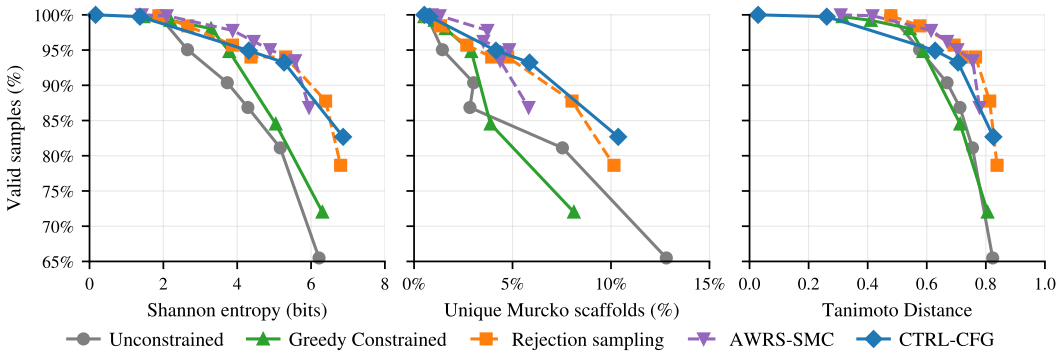

Figure 1: **Molecule validity and diversity.** Each curve sweeps sampling temperature from 0.5 to 2.5, averaging results over three molecule classes with 256 samples per class. CTRL-CFG uses k=8, and AWRS-SMC uses five particles. Validity is the fraction of outputs accepted by RDKit. Diversity is computed among valid molecules. Higher values on both axes are better.

complete outputs and accepts those satisfying the grammar. This provides a reference accuracy, as rejection sampling draws from the exact constrained distribution. Lastly, we compare with sequential Monte Carlo (SMC) methods: AWRS-SMC (Lipkin et al., 2025) for the molecule generation benchmark and P-GCD (Dang et al., 2026) for the text-to-SQL benchmark. P-GCD uses CTRL-G as a proposal distribution, and hence lies closest to our work.

6.2 Molecule Generation

Task and setup. Similar to Parys et al. (2025), we sample molecules in the SMILES grammar for three molecule classes (acrylates, chain extenders, and isocyanates). We distill a separate HMM ($h = 512$) from Gemma 4 E2B-it for each class on 90k LLM samples, and generate 256 samples per class and decoding configuration. Results are averaged across the three classes.

Metrics. Grammatical SMILES strings do not necessarily represent chemically valid molecules, so we measure molecule validity using RDKit (Landrum et al., 2025). We also measure sample diversity among valid molecules: Shannon entropy over canonical SMILES, mean pairwise Tanimoto distance between Morgan-3 fingerprints, and the fraction of unique Murcko scaffolds. The latter two capture molecular dissimilarity and structural diversity, respectively.

Validity and diversity. Figure 1 shows that CTRL-CFG achieves higher validity than unconstrained sampling and LCD as diversity increases. The improvement is visible across all three diversity metrics, and CTRL-CFG approaches the performance of rejection sampling. Thus, a short look-ahead window ($k=8$) recovers much of the sample quality lost through local masking on this task.

Runtime comparison. Figure 2 compares the speed–validity trade-off of the different methods at iso-diversity. Specifically, we tuned the sampling temperature of each method to yield about 6 bits of entropy. At k=8, CTRL-CFG achieves validity close to rejection sampling at higher throughput, and improves both validity and throughput over the measured AWRS-SMC configurations. We also ablate the look-ahead size k. As expected, the sample quality increases for longer look-ahead, at the cost of decreased throughput. To compare look-ahead with spending compute on a larger LM, we also evaluate LCD with the larger Gemma 4 E4B-it model. CTRL-CFG with E2B-it and k=4 achieves similar validity and entropy with 8.3% higher throughput compared to the E4B-it model.

6.3 Regular-Expression Synthesis

Task and setup. We evaluate Gemma 4 E2B-it on the 996 test annotations of StructuredRegex (Ye et al., 2020), which requires generating programs in a regular-expression DSL. Each prompt contains six training demonstrations, and we sample at temperature 1 with a 200-token budget. CTRL-CFG uses k=8 and HMMs with $h \in \{1024, 4096\}$, distilled from one million samples of the base model.

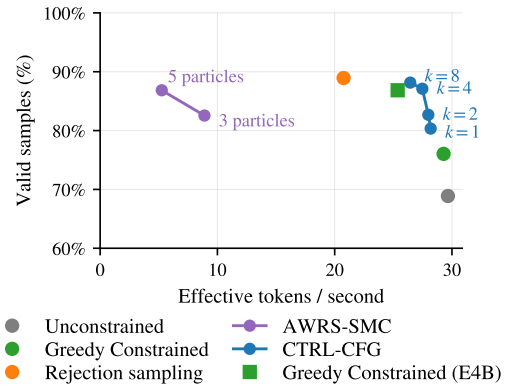

Figure 2: **Molecule validity and decoding throughput.** Temperatures are chosen to give approximately six bits of empirical entropy. CTRL-CFG varies $k \in \{1, 2, 4, 8\}$, and AWRS-SMC uses three or five particles. Throughput counts returned tokens, including the cost of discarded samples. Rejection throughput is estimated from unconstrained throughput and the acceptance rate (see Appendix D.1).

Method	Accuracy (%) ↑	I/O accuracy (%) ↑	KL (nats) ↓
Unconstrained	5.15 ± 0.21	6.06 ± 0.25	0.00 ± 0.00
LCD (XGrammar)	6.59 ± 0.12	9.34 ± 0.20	23.35 ± 0.62
CTRL-CFG (h=1024)	7.10 ± 0.12	10.01 ± 0.46	19.66 ± 0.61
CTRL-CFG (h=4096)	7.20 ± 0.31	10.44 ± 0.17	18.96 ± 0.58

Table 1: **StructuredRegex synthesis.** Accuracies report mean $\pm$ sample standard deviation over 3 runs on the test set ($n = 996$). Accuracy measures semantic equivalence, while I/O accuracy measures agreement with the input-output examples. KL reports mean cumulative next-token divergence per generation $\pm$ approximate 95% Monte Carlo half-width.

Metrics. We report semantic accuracy, measured by confirmed DFA equivalence with the reference program, and I/O accuracy, measured by consistency with the supplied examples. Table 1 reports accuracy means and sample standard deviations over three runs, together with cumulative next-token KL divergence to the base LM (Appendix D.2).

Results. CTRL-CFG yields modest improvements in both semantic and I/O accuracy over LCD in the reported runs. With 4096 HMM states, accuracy increases from 6.59% to 7.20%, while I/O accuracy increases from 9.34% to 10.44%. Using a larger HMM is beneficial as it provides a more faithful approximation to the base model, although the absolute accuracy differences are small.

6.4 Text-to-SQL Generation

Task and setup. We use Llama 3.1 8B-Instruct on the Spider (Yu et al., 2018) text-to-SQL benchmark. We use the same HMM, prompts, sampling settings, and 500-example test set as P-GCD (Dang et al., 2026). P-GCD only handles regular languages, and hence approximates the SQL grammar with a limited-depth NFA. We instead directly use an LL(1) SQL grammar. We also demonstrate the use of both P-GCD and CTRL-CFG as proposals for SMC. Both SMC variants use 3 particles and multinomial resampling when the effective sample size falls below 1.5.

Metrics. Execution accuracy measures agreement with the reference query's result on the database. We report the mean and sample standard deviation over three seeds. For SMC, accuracy is averaged using the final particle weights. We also measure decoding time per returned token, peak GPU memory, and sequence-level KL divergence to the base model.

Accuracy and runtime. Using the same grammar, CTRL-CFG increases mean execution accuracy from 62.18% to 63.52%, at 40.59 rather than 26.11 ms/token (Table 2). Using CTRL-CFG within SMC further increases mean accuracy to 66.27%, slightly exceeding rejection sampling. Our

Constraint	Method	Accuracy ↑ (%)	KL ↓ (nats)	Time ↓ (ms/token)	Memory ↓ (GB)
None	Unconstrained	62.99 ± 1.45	0.00 ± 0.00	25.44	16.37
NFA	LCD	63.12 ± 1.45	3.84 ± 0.57	26.02	16.37
	P-GCD	59.91 ± 0.31	3.54 ± 0.47	148.97	39.35
	P-GCD + SMC†	65.45 ± 1.13	–	471.47	40.67
CFG	LCD	62.18 ± 1.18	1.45 ± 0.26	26.11	16.38
	CTRL-CFG	63.52 ± 1.29	1.32 ± 0.22	40.59	33.75
	CTRL-CFG + SMC†	66.27 ± 1.17	–	81.09	34.79
	Rejection sampling	65.93 ± 0.91	0.29 ± 0.02	136.17	16.37

Table 2: **Spider execution accuracy, distributional change, and decoding cost.** Accuracy reports mean $\pm$ standard deviation over 3 runs. KL divergence reports mean $\pm$ approximate 95% Monte Carlo half-width to the base LLM distribution. Inference speed and peak GPU memory usage are measured on an L40S GPU. (†) SMC uses three particles with accuracy averaged using final particle weights. Full details are in Appendix D.3.

P-GCD reproduction reaches 59.91% at 148.97 ms/token, which increases to 65.45% with 3 particles. P-GCD incurs more overhead as the grammar needs to be unrolled into a finite state machine. CTRL-CFG has an estimated KL of 1.32 nats per sequence, compared with 1.45 for LCD under the same CFG. The NFA methods have larger estimates: 3.54 for P-GCD and 3.84 for LCD. Rejection sampling achieves the lowest divergence from the base model at 0.29 nats per sequence.

7 Limitations

Broader applicability of CTRL-CFG will require handling grammars beyond LL(1) and reducing the memory footprint of the HMM marginalization for large grammars. Moreover, tractable constrained decoding approaches such as CTRL-CFG require the LLM and grammar vocabulary to align. This work does not present a general solution towards this, and our experiments used grammars that are manually aligned to the LLM token vocabulary. Potential solutions for future work include byte-level models (Xue et al., 2022), or composing the grammar with a transducer (Snæbjarnarson et al., 2026). The modest gains on some tasks also motivate further investigation of surrogate quality, building on improvements to HMMs such as Yidou-Weng et al. (2025).

8 Conclusion

We extended the tractable-surrogate approach to constrained decoding from regular languages to context-free grammars. As constrained decoding with HMMs is cubic in the sequence length, we proposed a linear-time *look-ahead window* approximation for LL(1) grammars. Experiments on program synthesis and molecule generation illustrate its practical potential and the trade-offs that remain. This work takes a step towards more broadly usable tractable constrained decoding, even though vocabulary alignment and surrogate quality remain important directions for further work.

AI use statement

In this work, we used generative AI tools for implementing methods, helping develop theoretical models or conceptual frameworks, proposing or refining hypotheses, and assisting in the writing of proofs. We have not used generative AI tools for assisting with translation, formulating mathematical claims, providing critical ingredients for proving mathematical claims, designing or providing feedback on research methodology or experiments, or interpreting results. Generating synthetic data sets, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for brainstorming, and sourcing/searching for information. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. Controllable generation via locally constrained resampling. *Neurosymbolic AI: Foundations and Applications*, pp. 159–182, 2026.

Antoine Amarilli, Mikaël Monet, Paul Raphaël, and Sylvain Salvati. On the complexity of language membership for probabilistic words. *CoRR*, abs/2510.08127, 2025. doi: 10.48550/ARXIV.2510.08127.

Annaëlle Baiget, Jaron Maene, Seongmin Lee, Benjie Wang, Guy Van den Broeck, and Miryung Kim. Explainfuzz: Explainable and constraint-conditioned test generation with probabilistic circuits. *arXiv preprint arXiv:2604.06559*, 2026.

Alberto Bertoni, Massimiliano Goldwurm, and Nicoletta Sabadini. The complexity of computing the number of strings of given length in context-free languages. *Theoretical Computer Science*, 86(2):325–342, 1991.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding llms the right way: fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 3658–3673, 2024.

Y Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, 6, 2020.

Meihua Dang, Linxin Song, Honghua Zhang, Jieyu Zhao, Guy Van den Broeck, and Stefano Ermon. Mitigating bias in locally constrained decoding via tractable proposals. In *Forty-third International Conference on Machine Learning*, 2026.

Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, 2003.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *Proceedings of Machine Learning and Systems*, 7, 2025.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. Jsonschemabench: A rigorous benchmark of structured outputs for language models. *arXiv preprint arXiv:2501.10868*, 2025.

Emmanuel Anaya Gonzalez, Sairam Vaidya, Kanghee Park, Ruyi Ji, Taylor Berg-Kirkpatrick, and Loris D'Antoni. Constrained sampling for language models should be easy: An mcmc perspective. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Google. Gemma 4: Byte for byte, the most capable open models — blog.google. `https://blog.google/innovation-and-ai/technology/developers-tools/gemma-4/`, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1535–1546, 2017.

Richard M Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of algorithms*, 10(3):429–448, 1989.

Greg Landrum, Paolo Tosco, Brian Kelley, Ricardo Rodriguez, David Cosgrove, Riccardo Vianello, sriniker, Peter Gedeck, Gareth Jones, Eisuke Kawashima, NadineSchneider, Dan Nealschneider, Andrew Dalke, tadhurst cdd, Matt Swain, Brian Cole, Samo Turk, Aleksandr Savelev, Alain Vaucher, Maciej Wójcikowski, Ichiru Take, Hussein Faara, Vincent F. Scalfani, Rachel Walker, Daniel Probst, Kazuya Ujihara, Niels Maeder, Jeremy Monat, Juuso Lehtivarjo, and guillaume godin. rdkit/rdkit: 2025_03_5 (q1 2025) release, July 2025.

Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash Mansinghka. Sequential monte carlo steering of large language models using probabilistic programs. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*, 2023.

Ben Lipkin, Benjamin LeBrun, Jacob Hoover Vigly, João Loula, David R. MacIver, Li Du, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Timothy J. O'Donnell, Alexander K. Lew, and Tim Vieira. Fast controlled generation from language models with adaptive weighted rejection sampling. In *Second Conference on Language Modeling*, 2025.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, et al. Syntactic and semantic control of large language models via sequential monte carlo. In *International Conference on Learning Representations*, volume 2025, pp. 64758–64791, 2025.

Albert Lu, Hongxin Zhang, Yanzhe Zhang, Xuezhi Wang, and Diyi Yang. Bounding the capabilities of large language models in open text generation with prompt constraints. In *Findings of the Association for Computational Linguistics: EACL 2023*, pp. 1982–2008, 2023.

Jaron Maene and Luc De Raedt. The gradient of algebraic model counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19367–19377, 2025.

Reda Marzouk and Colin de La Higuera. Marginal inference queries in hidden markov models under context-free grammar constraints. *arXiv preprint arXiv:2206.12862*, 2022.

Daniel Melcer, Sujan Kumar Gonugondla, Pramuditha Perera, Haifeng Qian, Wen-Hao Chiang, Yanjun Wang, Nihal Jain, Pranav Garg, Xiaofei Ma, and Anoop Deoras. Approximately aligned decoding. *Advances in Neural Information Processing Systems*, 38:11445–11479, 2026.

William C Merrill. *A theory of the computational power and limitations of language modeling architectures*. PhD thesis, New York University, 2025.

Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. Cgmh: Constrained sentence generation by metropolis-hastings sampling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 6834–6842, 2019.

Niels Mündler, Jasper Dekoninck, and Martin Vechev. Constrained decoding of diffusion LLMs with context-free grammars. In *NeurIPS 2025 Fourth Workshop on Deep Learning for Code*, 2025.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. Grammar-aligned decoding. *Advances in Neural Information Processing Systems*, 37:24547–24568, 2024.

Kanghee Park, Timothy Zhou, and Loris D'Antoni. Flexible and efficient grammar-constrained decoding. In *International Conference on Machine Learning*, pp. 48262–48275. PMLR, 2025.

PaweĹ Parys, Sairam Vaidya, Taylor Berg-Kirkpatrick, and Loris D'Antoni. Constrained adaptive rejection sampling. *arXiv preprint arXiv:2510.01902*, 2025.

Clemente Pasti, Andreas Opedal, Timothy J. O'Donnell, Ryan Cotterell, and Tim Vieira. Prefix parsing is just parsing. In *Proceedings of ACL*, 2026.

Lianhui Qin, Sean Welleck, Daniel Khashabi, and Yejin Choi. Cold decoding: Energy-based constrained text generation with langevin dynamics. *Advances in Neural Information Processing Systems*, 35:9538–9551, 2022.

L Rabiner and B Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1): 4–16, 1986.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.779.

Max Scribner, Antonio Vergari, and Vaishak Belle. Tractable hierarchical control of autoregressive language models. *arXiv preprint arXiv:2607.20483*, 2026.

Vésteinn Snæbjarnarson, Samuel Kiegeland, Tianyu Liu, Reda Boumasmoud, Ryan Cotterell, and Tim Vieira. Transducing language models. *arXiv preprint arXiv:2603.05193*, 2026.

Bas van Opheusden, Luigi Acerbi, and Wei Ji Ma. Unbiased and efficient log-likelihood estimation with inverse binomial sampling. *PLoS computational biology*, 16(12):e1008483, 2020.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Benchmarking multimodal regex synthesis with complex structures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6081–6094, 2020.

Gwen Yidou-Weng, Ian Li, Anji Liu, Oliver Broadrick, Yuchen Cui, Guy Van den Broeck, and Benjie Wang. Learning tractable distributions of language model continuations. *arXiv preprint arXiv:2511.16054*, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 3911–3921, 2018.

Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. Tractable control for autoregressive language generation. In *International Conference on Machine Learning*, pp. 40932–40945. PMLR, 2023.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. Adaptable logical control for large language models. *Advances in Neural Information Processing Systems*, 37:115563–115587, 2024.

Honghua Zhang, Benjie Wang, Marcelo Arenas, and Guy Van den Broeck. Restructuring tractable probabilistic circuits. In *International Conference on Artificial Intelligence and Statistics*, pp. 2566–2574. PMLR, 2025.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael Littman, and Stephen Bach. Planetarium: A rigorous benchmark for translating text to structured planning languages. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11223–11240, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.560.

A Grammar Transformations

To convert an arbitrary LL(1) grammar into BNF (Definition 2), we apply the following transformations. The resulting BNF has a linear overhead in the input grammar size, and preserves the LL(1) and unambiguity properties when present. First, we remove unproductive and unreachable nonterminals and their productions. Second, for generation, append a fresh end-of-sequence token using a new start rule $S_{\mathsf{eos}} \to S\mathsf{eos}$. Third, replace each terminal a in a right-hand side of length at least two by a fresh nonterminal $T_a \to a$. Fourth, replace each rule $A \to B_1 \cdots B_n$ with $n \geq 3$ by

$$A \to B_1 C_1, \quad C_1 \to B_2 C_2, \quad \ldots, \quad C_{n-2} \to B_{n-1} B_n,$$

using a fresh C_i for each rule. Each new nonterminal has one production, and the original alternatives retain their first sets and nullability, so parsing decisions are unchanged. Empty productions remain in the grammar.

B Algorithms

B.1 Shared HMM chart

Algorithm 2 contains the pseudo-code for exact marginalization. We abbreviate the probability matrices as

$$W_A(\ell)[z, z'] = p_{\mathsf{HMM}}\big(A \overset{*}{\Rightarrow}_G x_{0:\ell},\, Z_\ell = z' \mid Z_0 = z\big),$$

and write $W_a[z, z'] = p_{\mathsf{HMM}}(a \mid z) p_{\mathsf{HMM}}(z' \mid z)$ for a token's emission–transition matrix. At each length, process B before A for $A \to B$. For $A \to BC$, process B first if C is nullable, and C first if B is nullable. This dependency graph is acyclic: a cycle would yield a nontrivial derivation $A \overset{*}{\Rightarrow}_G A$ by replacing siblings with ϵ, contradicting unambiguity in a reduced grammar.

Algorithm 2 Exact CFG Marginalization over an HMM.

Require: Unambiguous grammar $G = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ in BNF,
Require: HMM emission–transition matrices $W_a \in \mathbb{R}^{h \times h}$ as defined in Appendix B.1,
Require: sequence length $n \geq 1$.

Compute nullability and the dependency order from Appendix B.1.
for $A \in \mathcal{N}$ **do**
    $W_A(0) \leftarrow [\![A \overset{*}{\Rightarrow}_G \epsilon]\!] I_h$
    for $\ell \in \{1, \ldots, n\}$ **do**
        $W_A(\ell) \leftarrow 0_{h \times h}$
for $(A \to a) \in \mathcal{R}$ **do**
    $W_A(1)$ += W_a
for $\ell \in \{1,\ 2,\ \ldots,\ n\}$ **do**
    for $A \in \mathcal{N}$ in dependency order (children before parents) **do**
        for $(A \to B) \in \mathcal{R}$ **do**
            $W_A(\ell)$ += $W_B(\ell)$
        for $(A \to BC) \in \mathcal{R}$ **do**
            $W_A(\ell)$ += $\sum_{j=0}^{\ell} W_B(j) W_C(\ell - j)$
return $\{W_A(\ell) \mid A \in \mathcal{N},\ 0 \leq \ell \leq n\}$

B.2 Cached parser-stack probabilities

For the LL(1) grammar G, we assume a cache of the base-grammar matrices $W_A(\ell)$ and prefix vectors $b_A(\ell) = W_{A'}(\ell)\mathbf{1}$. Primed symbols occur only as unary or right children, so their vectors can be computed directly by marginalizing the final state in Algorithm 2. For a productive stack suffix β, define

$$f_\beta(\ell)[z] = p_{\mathsf{HMM}}\big(x_{0:\ell} \in \mathsf{L}(\mathrm{Pre}(G_\beta)) \mid z_0 = z\big).$$

Algorithm 3 implements Equation 5 using linked suffix nodes, with a jump pointer that skips symbols deriving only ϵ. The last window token lies in the pushed symbol or in its suffix. LL(1) makes

these cases disjoint, and the split $j = 0$ handles nullability. During parsing, push right-hand sides in reverse order and pop lookahead-selected empty derivations without expanding their subtrees. With $q_t[z] = p_{\mathsf{HMM}}(z_t = z \mid x_{0:t})$, the window probability is $q_t f_\beta(k)$ by Equation 4.

To simplify the exposition, Algorithms 3 and 4 do not handle termination before the end of the look-ahead window. However, this straightforward modification is done in our implementation.

Algorithm 3 Cached window probabilities on parser-stack suffixes.

Require: Matrices $W_A(\ell)$, prefix vectors $b_A(\ell)$, and window size $k \geq 1$.
Require: For each A, whether it derives a non-empty terminal string.
Initialize an empty suffix node with $f_\epsilon(0) = \mathbf{1}$ and $f_\epsilon(\ell) = \mathbf{0}$ for $1 \leq \ell \leq k$.
$\mathsf{jump}(\epsilon) \leftarrow \epsilon$
function PUSH(A, β)
    Create a node $\gamma = A\beta$ pointing to the cached suffix β.
    $f_\gamma(0) \leftarrow \mathbf{1}$
    for $\ell = 1, \ldots, k$ **do**
        $f_\gamma(\ell) \leftarrow b_A(\ell) + \sum_{j=0}^{\ell-1} W_A(j) f_\beta(\ell - j)$
    if A derives a non-empty terminal string **then**
        $\mathsf{jump}(\gamma) \leftarrow \gamma$
    else
        $\mathsf{jump}(\gamma) \leftarrow \mathsf{jump}(\beta)$
    return γ
function POP($A\beta$)
    Discard the top node and return its cached suffix β.
function WINDOWPROBABILITY(β, q)
    return $q f_\beta(k)$
Initialize the parser stack $\beta \leftarrow$ PUSH(S, ϵ).

B.3 ALL NEXT-TOKEN MARGINALS

Algorithm 4 computes the membership probability for every possible next token in one backward pass. For a nonterminal A that generates the first j tokens of the window, the vector $v_A(j)$ stores the weights for the remaining part of the window, indexed by the HMM state after those j tokens. These weights multiply the cached matrix $W_A(j)$ on the right.

The algorithm first initializes these vectors from the cached parser stack. If the window ends within A, there is no remaining continuation to check, giving the contribution $\mathbf{1}$ to $v_{A'}(k)$. If A generates $j < k$ tokens, the suffix δ contributes its cached vector $f_\delta(k - j)$. Only leading nullable symbols and the first nonnullable symbol can generate the next token. Jump pointers skip symbols deriving only ϵ, and LL(1) gives the visited symbols disjoint terminal first sets, limiting their number to $|\mathcal{N}|$.

The backward pass then follows the grammar rules to obtain the weights for each possible first token. For a binary rule $A \to BC$, it combines the weights after A with the cached probabilities for C to update B. If B is nullable, the first token can instead come from C. Vectors for primed symbols have equal entries throughout this computation, so a primed right child D' needs only the cached vector b_D, multiplied by that common entry.

The terminal-rule updates compute $g_t(a)$, the joint probability of next token a and window membership. Dividing by the HMM next-token probability $\pi_t(a) = p_{\mathsf{HMM}}(x_t = a \mid x_{0:t})$ gives the conditional membership probability $u_t(a)$ in Equation 6.

Algorithm 4 All next-token membership scores.

Require: Rules and dependency order of $\mathrm{Pre}(G)$, base matrices W, and prefix vectors b.
Require: Cached stack β with jump pointers, HMM state row q_t, and window size $k \geq 1$.
$v_A(j) \leftarrow \mathbf{0}$ for every original or primed symbol A and $1 \leq j \leq k$.
$g_t(a) \leftarrow 0$ for each $a \in \Sigma$
$\eta \leftarrow \mathsf{jump}(\beta)$
while $\eta \neq \epsilon$ **do**
  $A \leftarrow \mathsf{top}(\eta)$ and $\delta \leftarrow \mathsf{tail}(\eta)$ ▷ Read without popping
  $v_{A'}(k) \mathrel{+}= \mathbf{1}$
  for $j = 1, \ldots, k-1$ **do**
    $v_A(j) \mathrel{+}= f_\delta(k-j)$
  if A is not nullable **then**
    break
  $\eta \leftarrow \mathsf{jump}(\delta)$
for $j = k, k-1, \ldots, 1$ **do**
  for each original or primed A in reverse dependency order **do**
    for each rule $A \to B$ **do**
      $v_B(j) \mathrel{+}= v_A(j)$
    for each rule $A \to BC$ **do**
      for $s = 1, \ldots, j$ **do**
        if $C = D'$ is primed **then**
          $v_B(s) \mathrel{+}= b_D(j-s) v_A(j)[1]$
        else
          $v_B(s) \mathrel{+}= W_C(j-s) v_A(j)$
      if B is nullable **then**
        $v_C(j) \mathrel{+}= v_A(j)$
for each terminal rule $A \to a$ in $\mathrm{Pre}(G)$ **do**
  $g_t(a) \mathrel{+}= q_t W_a v_A(1)$
return $u_t(a) = g_t(a)/\pi_t(a)$ for $\pi_t(a) > 0$, and 0 otherwise.

C Proofs

C.1 Proof of Theorem 3

Proof. We prove the theorem with a reduction from #DNF, which is #P-hard under polynomial-time Turing reductions (Karp et al., 1989). Let $\varphi = T_1 \vee \cdots \vee T_m$ be a DNF with $k \geq 1$ variables (assuming no contradictory terms). Let $D_{i,j} \subseteq \{0,1\}$ denote the possible assignments to the jth variable in the ith term. We now construct a grammar over the vocabulary $\Sigma = \{0, 1, c_1, \ldots, c_m, \mathsf{eos}\}$, with the following rules

$$S \to A_{i,0}, \qquad A_{i,j} \to b A_{i,j+1} \text{ (if } b \in D_{i,j}), \qquad A_{i,k} \to c_i \mathsf{eos},$$

for $1 \leq i \leq m$ and $0 \leq j < k$. Terminal extraction and binarization yield a reduced acyclic BNF grammar of size $O(mk)$. Its sequences are $a c_i \mathsf{eos}$ with $a \models T_i$. The marker c_i selects a term and the bits select a unique derivation within it, so the grammar is unambiguous even when an assignment satisfies several terms. Its length-k prefixes are exactly the satisfying assignments of φ. A one-state HMM emitting uniformly over $\Sigma \setminus \{\mathsf{eos}\}$ therefore gives

$$p_{\mathsf{HMM}}(x_{0:k} \in \mathsf{PL}(G)) = \frac{\#\mathsf{DNF}(\varphi)}{(m+2)^k}.$$

The construction, unary window length, and scaling factor (with $O(k \log(m+2))$ bits) all have polynomial size. □

C.2 Proof of Theorem 4

Proof. In a parse tree for a non-empty w rooted at S' in $\mathrm{Pre}(G)$, the primed symbols form a path to the last token of w. Subtrees to its left use only rules of G. Choose an inducing original rule at

each primed node, and restore any omitted right siblings using productive derivations. This extends the tree to a G-parse of wy, possibly with $y = \epsilon$. Conversely, truncate such a complete tree at the last token of w: prime the path to that token and discard subtrees to its right. At a binary node, this gives $A' \to B'$ or $A' \to BC'$ according to which child contains the token. Truncating any extension constructed above recovers its original prefix tree.

Suppose for contradiction that a non-empty w has two distinct prefix parse trees. Extend them as above to G-parse trees of wy_1 and wy_2. Run the LL(1) parser on both completions, stopping immediately after consuming w and before inspecting the next token. The runs must agree up to that point: they start with the same stack, and each parsing decision uses the same next token of w as lookahead, including decisions to derive ϵ. Their truncated trees are therefore identical, but these are precisely the two original prefix trees, a contradiction. Finally, primed symbols are nonnullable, so $\hat{S} \to \epsilon$ is the unique derivation of the empty string. This also covers complete words ending in EOS and proves unambiguity. □

C.3 Proof of Theorem 5

Proof. Chart precomputation takes $O(k^2h^3|\mathcal{R}|)$ time and $O(kh^2|\mathcal{N}|)$ space. The prefix transform, nullability, and dependency ordering have linear overhead. Nullable endpoint splits require only matrix additions.

To bound parser pushes, consider the expansions before consuming one token. Empty derivations are popped directly. Every other expansion lies on the ancestor path to that token. A repeated nonterminal on this path would imply $A \overset{*}{\Rightarrow}_G A\gamma$ through nullable left siblings, contradicting LL(1). The path therefore contains at most $|\mathcal{N}|$ nonterminals, each pushing at most two symbols. Including initialization, there are at most $1 + 2n|\mathcal{N}|$ pushes, and each node is popped at most once. The per-push bounds from the main text give $O(nk^2h^2|\mathcal{N}|)$ time and $O(nkh|\mathcal{N}|)$ live stack storage. Skipping empty subtrees is essential here: expanding them can take exponentially many steps in the grammar size.

Appendix B.3 initializes the backward pass from at most $|\mathcal{N}|$ stack symbols, then performs $O(k^2|\mathcal{R}|)$ matrix–vector updates per token. Nullable-child additions, terminal contractions, and HMM filtering fit within the resulting $O(k^2h^2|\mathcal{R}|)$ time bound. Its $O(kh|\mathcal{N}|)$ workspace is dominated by the chart. Summing these costs and using $|\mathcal{N}| \leq |\mathcal{R}|$ proves the theorem. The bounds exclude input model storage and LM evaluation. □

D Experimental Details

Grammar	Nonterminals	Productions	Token terminals
SMILES	31	141	94
StructuredRegex	1,350–1,386	101,574–103,242	59,951
SQL	701	100,780	97,539

Table 3: Sizes of the token-level LL(1) grammars before EOS augmentation and inference transformations. Each right-hand-side alternative counts as one production. StructuredRegex ranges cover the 27 grammars instantiated from constants in the test prompts.

Grammars. Table 3 counts the input grammars after tokenization and explicit left factoring. SMILES uses one grammar across the three molecule classes, while StructuredRegex specializes its literals to each prompt's constants. After EOS augmentation, binarization, prefix construction, and epsilon elimination, the SQL grammar has 2,119 nonterminals and 206,362 productions.

D.1 Molecule generation

HMM Distillation. Generating 90k training and 10k held-out responses per class took 5.6 hours across all three classes, excluding LM loading. The cost of training the HMM is negligible compared to sampling from the LLM.

Sampling. The temperature sweep uses $T \in \{0.5, 0.7, 1.0, 1.2, 1.3, 1.5, 2.0, 2.5\}$, with CTRL-CFG look-ahead k=8 and five particles for AWRS-SMC. Sampling temperature is applied to the reweighted logits. Shannon entropy is computed from the empirical frequencies of canonical SMILES among RDKit-valid outputs. Tanimoto distance uses radius-three Morgan fingerprints with 2048 bits, and scaffold diversity is the number of distinct Murcko scaffolds divided by the number of valid samples. Each metric is computed per class and then averaged across the three classes.

For Figure 2, CTRL-CFG uses T=1.4 and $k \in \{1, 2, 4, 8\}$. The E2B unconstrained, LCD, and rejection baselines use T=1.9. AWRS-SMC uses T=2.5 with three or five particles. E4B LCD uses T=1.6. These operating points give approximately six bits of empirical entropy. The throughput is measured on an L40S GPU, with batch size 1. Rejection sampling includes the time for all failed samples, divided by the number of tokens of the accepted sample. Similarly, AWRS-SMC includes the time for all particles.

D.2 Regular-expression synthesis

HMM Distillation. For each HMM size, we jointly train the HMM and its prefix-conditioning head on one million LM responses to original and conservatively augmented training prompts, selecting the checkpoint on unaugmented official development prompts. The HMM prior is refreshed at every generated token (Yidou-Weng et al., 2025).

Generating the shared training set and 17,550 development responses with prefix features took about 19 hours on two L40S and two RTX A5000 GPUs (38 GPU-hours per type). Joint fitting from the existing HMM and head initialization took 0.77 and 2.36 L40S GPU-hours for h=1024 and 4096, respectively, over two epochs (157.6 million suffix-token presentations).

Sampling. All reported runs use the same prompt, and one sampled response per annotation (beam size 1). All methods sample with temperature 1 and max 200 new tokens (excluding the prompt). When the remaining length fits within the k+1-token window, the upper bound of Equation 3 is replaced by exact HMM marginalization.

KL estimation. We compute the exact full-vocabulary next-token divergence from the decoding policy to the raw temperature-1 base LM at every visited prefix. We sum until generation finishes (either by EOS or the max token limit), and average over all 996 annotations and three seeds. The resulting statistic measures cumulative decoding-policy divergence, not KL between distributions of completed outputs or conditional on successful completion. Approximate 95% Monte Carlo half-widths are 1.96 times the standard error estimated from within-annotation variation across seeds, keeping the annotation set fixed. Unconstrained sampling has zero divergence by definition.

D.3 Text-to-SQL generation

Constraints and sampling. All Spider runs use the same 500 test examples used by Dang et al. (2026). We also reuse their distilled HMM, with no prompt conditioning. Execution accuracy uses the external Spider evaluator's default settings. All CFG methods in Table 2, including rejection sampling, use the same LL(1) grammar. The NFA constraint uses the token automaton of Dang et al. (2026).

Sampling uses temperature 1, with a minimum of 5 and a maximum of 128 new tokens, and no top-k or top-p truncation. Rejection sampling draws unconstrained candidates until the first complete sequence is accepted, without a retry cap. CTRL-CFG uses a k=8 look-ahead, and P-GCD uses constraint weight c=0.1. Both SMC variants use three particles and LM/proposal importance correction. Multinomial resampling with replacement occurs when the effective sample size is below $0.5 \times 3 = 1.5$. The reported accuracy averages correctness using final particle weights. Selecting the highest-weight particle of SMC + CTRL-CFG gives $67.00 \pm 2.17\%$, and the unweighted particle mean is $64.30 \pm 1.18\%$.

KL estimation. For LCD and single-particle CTRL-CFG and P-GCD, we compute the exact next-token divergence between the sampling distribution Q and the raw temperature-1 LM distribution P over the full vocabulary at each visited prefix. We sum these divergences along each generated sequence and average over the three seeds. The reference uses the same EOS token and 128-token stopping horizon, but no grammar or minimum-length mask. (The reported KL therefore includes

the sampler's minimum-length EOS restriction.) Approximate 95% Monte Carlo half-widths are 1.96 times the standard error estimated from within-question variation across seeds.

For rejection sampling, let R be the candidate sampler with the minimum-length rule and A the event of grammar acceptance. The returned distribution is $Q = R(\cdot \mid A)$, giving $D_{\mathrm{KL}}(Q\|P) = -\log R(A) + \mathbb{E}_Q[\log(R(X)/P(X))]$. For the saved count T through first acceptance, the harmonic number $H_{T-1} = \sum_{n=1}^{T-1} 1/n$ is an unbiased estimate of $-\log R(A)$ by inverse binomial sampling (van Opheusden et al., 2020).

Resource measurements. GPU memory is peak allocated memory in decimal GB, including model weights and all particles for SMC. Time is average generation wall time per returned token, including prefill and EOS but excluding setup and evaluation. Measurements use an NVIDIA L40S and one prompt at a time. Unconstrained sampling, LCD, and CTRL-CFG use the full 500-question runs. SMC runtime includes all particles' work, divided by tokens in the highest-weight outputs. Rejection sampling similarly includes the work of all rejected samples, divided by the tokens in the accepted output.