Tractable Control for Autoregressive Language Generation

Honghua Zhang [*1] **Meihua Dang** [*1] **Nanyun Peng** [1] **Guy Van den Broeck** [1]

Abstract

Despite the success of autoregressive large language models in text generation, it remains a major challenge to generate text that satisfies complex constraints: sampling from the conditional distribution $\Pr(\text{text} \mid \alpha)$ is intractable for even the simplest lexical constraints α. To overcome this challenge, we propose to use tractable probabilistic models (TPMs) to impose lexical constraints in autoregressive text generation models, which we refer to as **GeLaTo** (**Ge**nerating **La**nguage with **T**ractable C**o**nstraints). To demonstrate the effectiveness of this framework, we use distilled hidden Markov models, where we *can* efficiently compute $\Pr(\text{text} \mid \alpha)$, to guide autoregressive generation from GPT2. GeLaTo achieves state-of-the-art performance on challenging benchmarks for constrained text generation (e.g., CommonGen), beating various strong baselines by a large margin. Our work not only opens up new avenues for controlling large language models but also motivates the development of more expressive TPMs.

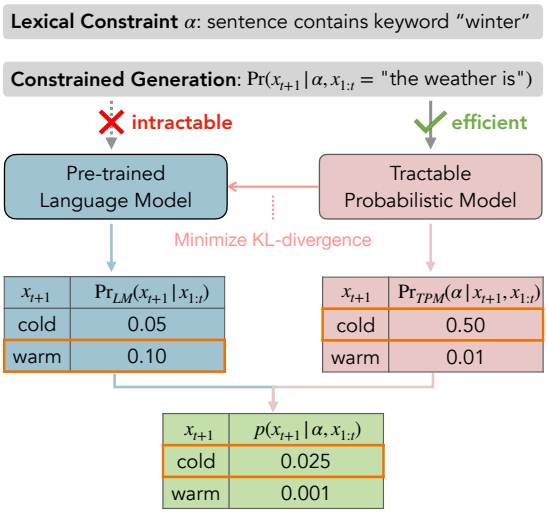

Figure 1. Given some lexical constraint α that we want our pre-trained language models to follow in generation, the conditional distribution $\Pr(x_{t+1} \mid x_{1:t}, \alpha)$ is often intractable. We propose to control and guide the autoregressive generation process of pre-trained LMs via tractable probabilistic models, which do support efficient computation of $\Pr(x_{t+1} \mid x_{1:t}, \alpha)$.

1. Introduction

Large pre-trained language models (LMs) (Radford et al., 2019; Lewis et al., 2020) have achieved remarkable performance on a wide range of challenging language generation tasks such as machine translation (Bahdanau et al., 2015; Luong et al., 2015), summarization (Liu et al., 2015; Xu & Durrett, 2019) and open-domain creative generation (Yao et al., 2019; Tian & Peng, 2022). Nevertheless, many practical language generation applications require fine-grained control of LMs to follow complex lexical constraints (e.g., given a source document, generate a summary that contains certain keywords). The common paradigm for controlling pre-trained LMs is to either finetune them on task-specific datasets or to condition them on certain prompts. However, fine-tuning and prompting are by nature approximate solutions and do not guarantee that the desired constraints are satisfied (Meng et al., 2022; Zhang et al., 2022). The major difficulty of constrained language generation lies in the autoregressive nature of LMs: they only model the next token distribution given some prefix $\Pr_{\text{LM}}(x_{t+1} \mid x_{1:t})$, while the conditional distribution $\Pr_{\text{LM}}(x_{1:n} \mid \alpha)$ given a constraint α as simple as, e.g., a keyword appearing at the end of a sentence, is often intractable (Roth, 1996).

Aside from language models based on neural architectures, one line of research in machine learning focuses on the development of *tractable probabilistic models* (TPMs) (Poon & Domingos, 2011; Kulesza & Taskar, 2012; Choi et al., 2020b; Zhang et al., 2021). TPMs model joint probability distributions and allow for efficient conditioning on various families of logical constraints (Kisa et al., 2014; Choi et al., 2015; Bekker et al., 2015). In this paper, we propose **GeLaTo** (**Ge**nerating **La**nguage with **T**ractable C**o**nstraints),

[*]Equal contribution [1]Department of Computer Science, University of California, Los Angeles, USA. Correspondence to: Honghua Zhang <hzhang19@cs.ucla.edu>, Meihua Dang <mhdang@cs.ucla.edu>, Nanyun Peng <violetpeng@cs.ucla.edu>, Guy Van den Broeck <guyvdb@cs.ucla.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

where we use TPMs to impose lexical constraints in autoregressive text generation. Given a pre-trained autoregressive LM $\Pr_{\text{LM}}$, e.g., GPT3 (Brown et al., 2020), our goal is to generate text effectively following the conditional distribution $\Pr_{\text{LM}}(x_{1:n} \mid \alpha)$ for arbitrary lexical constraints α. As illustrated in Figure 1, our proposed framework consists of two major components: (1) we train a TPM $\Pr_{\text{TPM}}$ via maximum likelihood estimation (MLE) on samples drawn from $\Pr_{\text{LM}}$, which is equivalent to minimizing the KL-divergence between $\Pr_{\text{TPM}}$ and $\Pr_{\text{LM}}$; then (2) at generation time, we compute $\Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha)$ efficiently and combine it with $\Pr_{\text{LM}}(x_{t+1} \mid x_{1:t})$ to approximate $\Pr_{\text{LM}}(x_{t+1} \mid x_{1:t}, \alpha)$ for reliable control. Note that we assume nothing about the lexical constraint α as we train $\Pr_{\text{TPM}}$, which means that the TPM does not need to be re-trained for different types of constraints: given a trained TPM that approximates $\Pr_{\text{LM}}$ well enough, we can use it to impose any lexical constraints α, as long as $\Pr_{\text{TPM}}(. \mid \alpha)$ can be efficiently computed.

Throughout this paper, we use hidden Markov models (HMMs) (Rabiner & Juang, 1986) as an example TPM to demonstrate the effectiveness of GeLaTo. Specifically, (1) we show that, when trained as probabilistic circuits (Choi et al., 2020b; Liu et al., 2023), HMMs can approximate the GPT2-large model fine-tuned on downstream tasks well enough and (2) we propose a dynamic programming algorithm that efficiently computes conditional probabilities $\Pr_{\text{HMM}}(\cdot \mid \alpha)$, for αs that encode constraints as conjunctive normal forms (CNFs):

$$(I(w_{1,1}) \vee \cdots \vee I(w_{1,d_1})) \wedge \cdots \wedge (I(w_{m,1}) \vee \cdots \vee I(w_{m,d_m}));$$

here each $w_{i,j}$ is *a string of tokens*, and $I(w_{i,j})$ is an indicator variable denoting whether or not w_{ij} appears in the generated text. Intuitively, constraint α requires that a set of m keywords must appear somewhere in the generated text, in any of their inflections, where each inflection is encoded as a string of one or more tokens. We evaluate the performance of GeLaTo on challenging constrained text generation datasets: CommonGen (Lin et al., 2020), News (Zhang et al., 2020), and Yelp!Review (Cho et al., 2019). GeLaTo not only achieves state-of-the-art generation quality but also guarantees that the constraints are satisfied 100%; for both unsupervised and supervised settings, GeLaTo beats strong baselines belonging to different families of constrained generation approaches by a large margin.

Our study demonstrates the potential of TPMs in controlling large language models and motivates the development of more expressive TPMs.

2. Guiding Autoregressive Generation with Tractable Probabilistic Models

In this section, we present the general GeLaTo framework for guiding autoregressive generation with tractable probabilistic models. Throughout this paper, we use uppercase letters X_t for random variables and lowercase letters x_t for their assignment.

Let $\Pr_{\text{LM}}(x_{1:n})$ be the distribution of an autoregressive LM (e.g., GPT) over n tokens and α a lexical constraint defined over $X_{1:n}$; our goal is to generate from the following conditional distribution:

$$\Pr_{\text{LM}}(x_{1:n} \mid \alpha) = \prod_t \Pr_{\text{LM}}(x_{t+1} \mid x_{1:t}, \alpha)$$

Though $\Pr_{\text{LM}}(x_{t+1} \mid x_{1:t}, \alpha)$ is intractable, we can assume that $\Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha)$ can be efficiently computed.

The first step of GeLaTo is to train our TPM model such that $\Pr_{\text{TPM}}$ approximates $\Pr_{\text{LM}}$ as well as possible. We train the TPM model via maximum likelihood estimation (MLE) on data drawn from $\Pr_{\text{LM}}$, that is, we maximize

$$\mathbb{E}_{x_{1:n} \sim \Pr_{\text{LM}}} \log \Pr_{\text{TPM}}(x_{1:n}),$$

which effectively minimizes their KL-divergence:

$$\begin{aligned} &D_{\text{KL}}(\Pr_{\text{LM}} \parallel \Pr_{\text{TPM}}) \\ &= \mathbb{E}_{x_{1:n} \sim \Pr_{\text{LM}}} \log \Pr_{\text{LM}}(x_{1:n}) - \mathbb{E}_{x_{1:n} \sim \Pr_{\text{LM}}} \log \Pr_{\text{TPM}}(x_{1:n}) \end{aligned}$$

With the recent development of scaling up TPMs (Chiu & Rush, 2020; Dang et al., 2022a; Liu et al., 2023), we show in Section 4 that it is possible to train TPMs as good enough approximations of LMs.

Now given some TPM as a good enough approximation for the LM that we want to generate from, we combine both models for constrained generation, where the TPM is responsible for providing guidance on incorporating lexical constraints and LM responsible for generating fluent texts. To derive our formulation, in addition to lexical constraint α, we assume that there exists some "quality" constraint β such that $\Pr_{\text{TPM}}(\mid \beta)$ is even closer to $\Pr_{\text{LM}}$; intuitively we interpret β as some constraint characterizing the high-quality (fluent & grammatical) sentences that are likely to be sampled from our base LM $\Pr_{\text{LM}}$. Hence, in order to generate a high-quality sentence satisfying some lexical constraint α, we generate from

$$\Pr_{\text{TPM}}(x_{1:n} \mid \alpha, \beta) = \prod_t \Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha, \beta);$$

in particular, in addition to the assumption that $\Pr_{\text{TPM}}(\cdot \mid \beta)$ is a good enough approximation for $\Pr_{\text{LM}}$, we also assume the *key independence assumption*: α and β are conditionally independent given $x_{1:t+1}$. By applying Bayes rule, it follows from our assumptions that:

$$\begin{aligned} &\Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha, \beta) \\ &\quad \propto \Pr_{\text{TPM}}(\alpha \mid x_{1:t+1}, \beta) \cdot \Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \beta) \\ &\quad \propto \Pr_{\text{TPM}}(\alpha \mid x_{1:t+1}) \cdot \Pr_{\text{LM}}(x_{t+1} \mid x_{1:t}). \end{aligned}$$

Now we examine whether our key independence assumption holds for the *unsupervised* and *supervised* settings.

Unsupervised setting. In the unsupervised setting, we assume that the base pre-trained LM is *not* fine-tuned given task-specific supervision; that is, $\Pr_{\text{LM}}$ is not fine-tuned to generate texts satisfying α provided as input, but is possibly fine-tuned or prompted for the purpose of domain adaptation. In this setting, there is no easy way for the "quality" constraint β to obtain any information about the lexical constraint α and our key independence assumption should roughly hold. In other words, satisfying the lexical constraint α should not help or hinder the fluency of the generated sentence according to the pre-trained LM, it merely biases what the sentence talks about. Hence for the unsupervised setting, we generate autoregressively following the next-token distribution defined as:

$$p(x_{t+1}|x_{1:t}, \alpha) \propto \Pr_{\text{TPM}}(\alpha|x_{1:t+1}) \cdot \Pr_{\text{LM}}(x_{t+1}|x_{1:t}). \quad (1)$$

This formulation is also adopted in FUDGE (Yang & Klein, 2021) and NADO (Meng et al., 2022), which train auxiliary models to approximate $\Pr_{\text{LM}}(\alpha \mid x_{1:t+1})$; the key difference is that such auxiliary models take α as input during training while our TPM training is unconditional.

Supervised setting. In this setting, we assume that the language model $\Pr_{\text{LM}}$ is fine-tuned in a sequence-to-sequence (seq2seq) manner; that is, during training, α is explicitly supplied to the LM together with some gold sentences: e.g., for keyword-type constraints, the LM is fine-tuned over texts of the form *"weather winter cold = the weather is cold in winter,"* where the prompt *"weather winter cold = "* encodes the constraint that all words before "=" should be used. In this case, our key independence assumption no longer holds because $\Pr_{\text{LM}}$ is already trained to satisfy the lexical constraint α, which is provided as part of the prefix $x_{1:t+1}$. Hence for the supervised setting, we adopt an alternative formulation by viewing $\Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha)$ and $\Pr_{\text{LM}}(x_{t+1} \mid x_{1:t})$ as classifiers trained for the same task yet with different biases; by Satopää et al. (2014), if we assume that each model predicts the true logits up to additive Gaussian noise, then the most likely logits can be found by taking a geometric mean of the models. Hence, in the supervised setting, we generate autoregressively following the next-token distribution defined as their weighted geometric mean (Hinton, 2002; Grover & Ermon, 2018):

$$p(x_{t+1} \mid x_{1:t}, \alpha) \propto \Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha)^{w} \cdot \Pr_{\text{LM}}(x_{t+1} \mid x_{1:t})^{1-w}; \quad (2)$$

here $w \in (0, 1)$ is a hyper-parameter to be tuned.

To summarize, GeLaTo consists of two major steps: (1) distillation: we train a TPM on samples drawn from the pre-trained LM via MLE to effectively minimize the KL divergence between $\Pr_{\text{LM}}$ and $\Pr_{\text{TPM}}$; (2) probabilistic reasoning: for each step of autoregressive generation, we compute $\Pr_{\text{TPM}}(\cdot \mid \alpha)$ and generate from the conditional next-token distribution $p(x_{t+1} \mid x_{1:t}, \alpha)$ defined above. In addition to better generation quality, which we demonstrate in Section 4, GeLaTo has two major advantages compared to its counterparts for constrained generation:

- The sentences generated following $p(x_{t+1} \mid x_{1:t}, \alpha)$ are *guaranteed* to satisfy the lexical constraint α; in autoregressive generation, as we generate the next token x_{t+1}, it follows from the definition that for choices of x_{t+1} such that α *cannot be satisfied*, $\Pr_{\text{TPM}}(x_{t+1}, x_{1:t}, \alpha)$ is 0, thus $p(x_{t+1} \mid x_{1:t}, \alpha)$ is also 0.
- The TPM training is independent of the lexical constraint α, which is only enforced at inference time; it immediately follows that we do not need to re-train the TPM model no matter how α changes; on the other hand, constrained decoding approaches that train auxiliary neural models, e.g., FUDGE and NADO, need to re-train their model for different types of constraints.

Throughout the rest of this paper, we use **hidden Markov models** (HMMs) as example TPMs to demonstrate the practicality and effectiveness of GeLaTo. In the following section, we propose an efficient algorithm for computing $\Pr_{\text{TPM}}(\alpha \mid x_{1:t+1})$ and $\Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha)$.

3. Efficient Probabilistic Reasoning with Hidden Markov Models

To impose lexical constraint α in autoregressive generation via TPM, for any given prefix $x_{1:t}$, we need to compute $\Pr_{\text{TPM}}(x_{1:t}, \alpha)$ (omit subscript for rest of the section); specifically, as described in Section 2, we need to compute $\Pr(\alpha \mid x_{1:t+1}) = \Pr(x_{1:t+1}, \alpha) / \Pr(x_{1:t+1})$ for the unsupervised setting and $\Pr(x_{t+1} \mid x_{1:t}, \alpha) \propto \Pr(x_{1:t+1}, \alpha)$ for the supervised setting. In this section, we describe a dynamic programming algorithm that computes $\Pr(x_{1:t}, \alpha)$ for hidden Markov models (HMMs), where α is some lexical constraint encoded in a conjunctive normal form (CNF):

$$(I(w_{1,1}) \vee \cdots \vee I(w_{1,d_1})) \wedge \cdots \wedge (I(w_{m,1}) \vee \cdots \vee I(w_{m,d_m}));$$

here each $w_{i,j}$ is a string of tokens, which we denote as "keystrings" for short, and $I(w_{ij})$ is the indicator variable that represents whether w_{ij} appears in the generated text.[1] We refer to $(I(w_{i,1}) \vee \cdots \vee I(w_{i,d_i}))$ as a *clause*.

For simplicity, we use the short-hand $\alpha_{l:r}$ to denote the event that α is satisfied on the *sub-sequence* $X_{l:r}$. In practice, we

[1] To be precise, denoting the kth token of w_{ij} as $(w_{ij})_k$, $I(w_{i,j})$ is in fact a disjunction over conjunctions: $\vee_{1 \le t \le n - |w_{ij}| + 1} \left(\wedge_{0 \le k < |w_{ij}|} X_{t+k} = (w_{ij})_k \right)$, representing that w_{ij} can be in any position of the generated text.

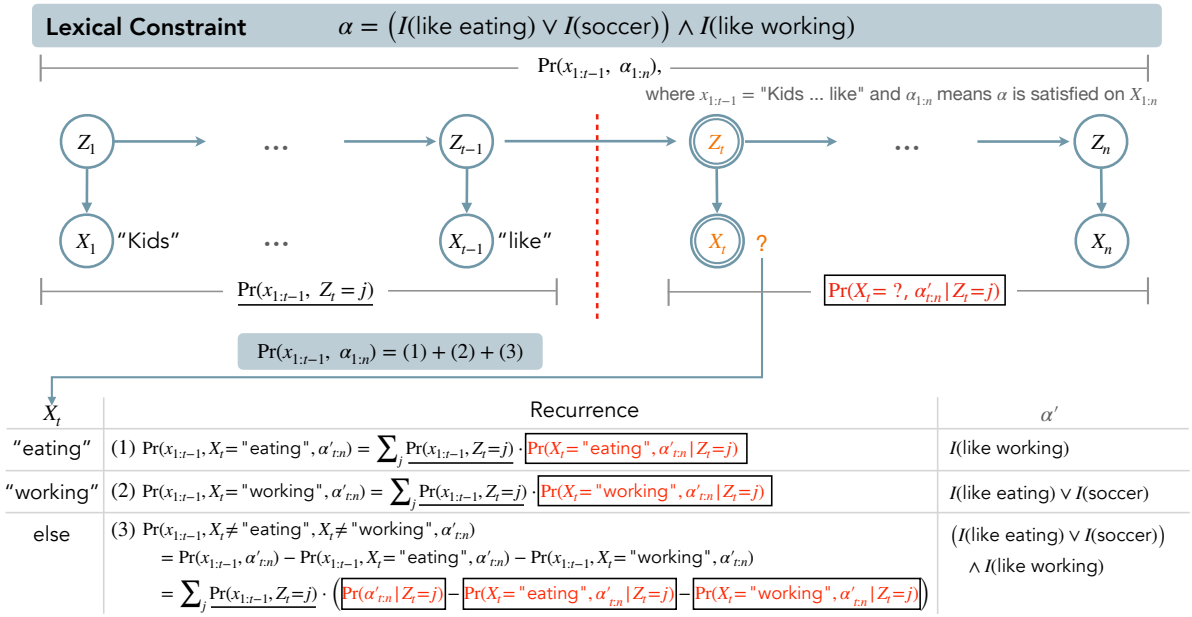

Figure 2. A toy example illustrating our dynamic programming algorithm. Here, given the the first $t-1$ tokens "Kids ... like" that have been generated, the figure illustrates how to compute $\Pr(X_{1:t-1} =$"Kids ... like", $\alpha_{1:n})$. We consider three possible cases for the next token X_t: "eating", "working" or neither, and for each case we can reduce the constraint $\alpha_{1:n}$ to the "easier" constraint $\alpha'_{t:n}$ for some α'. Then by conditioning on $Z_t = j$ (hidden states), we can break down $\Pr(x_{1:t-1}, X_t = ?, \alpha'_{t:n})$ into two terms: $\Pr(x_{1:t-1}, Z_t = j)$ and $\Pr(X_t = ?, \alpha'_{t:n} \mid Z_t = j)$, which are underlined and boxed in the figure, respectively; in particular the underlined terms can be computed by the forward algorithm for HMMs and the boxed terms can be computed recursively by the dynamic programming algorithm.

treat HMMs as language models over sequences of tokens of maximum length n and the lexical constraint we enforce is denoted as $\alpha_{1:n}$; in the following discussions, we write $\Pr(x_{1:t}, \alpha_{1:n})$ instead of $\Pr(x_{1:t}, \alpha)$.

3.1. Hidden Markov Models

A hidden Markov model (HMM) represents a joint probability distribution over n observed variables $X_{1:n}$ and n latent variables $Z_{1:n}$. Specifically, for language modeling, X_t represents the token at position t and Z_t represents the corresponding latent state; Z_t takes values in $\{1, 2, \ldots, h\}$, where h is the *number of latent states*. Given observed token sequence $x_{1:n}$ and latent state sequence $z_{1:n}$, the joint probability $\Pr(x_{1:n}, z_{1:n})$ is defined as:

$$\Pr(x_1 \mid z_1) \Pr(z_1) \prod_{2 \le t \le n} \Pr(x_t \mid z_t) \Pr(z_t \mid z_{t-1});$$

in particular, the parameters of HMM are given by the initial probability $\Pr(z_1)$, emission matrix $\Pr(x_t \mid z_t)$ and the transition matrix $\Pr(z_{t+1} \mid z_t)$, which stay the same across different positions t. HMMs can also be represented as Bayesian networks (Pearl, 1985); see Figure 2 for an example. To perform probabilistic inference on HMMs efficiently, we leverage the following *Markov property*:

$$\Pr(x_{t:n} \mid z_t, x_{1:t-1}) = \Pr(x_{t:n} \mid z_t). \tag{3}$$

For example, we can compute the probability of any prefix $\Pr(x_{1:t}) = \sum_{z_t} \Pr(x_{1:t}, z_t)$, which can be efficiently computed by the following recurrence relation, which is referred to as the *forward algorithm* (Rabiner & Juang, 1986):

$$\Pr(x_{1:t}, z_t) = \sum_{1 \le z_{t-1} \le h} \Pr(x_t \mid z_t) \Pr(z_t \mid z_{t-1}) \Pr(x_{t-1}, z_{t-1}).$$

Modeling Variable-length Texts with HMMs. HMMs model distributions over a fixed number of random variables $X_{1:n}$. To model texts with variable lengths, we first determine a maximum sequence length n and pad training texts of length $< n$ with the special EOS ("endoftext") token to the maximum length. We also construct our HMM in a special way such that an EOS token can only be followed by EOS tokens; that is, sequences that do not satisfy this constraint have 0 probability. Hence, $\Pr_{\text{HMM}}(x_{1:n})$ effectively defines a distribution over all texts with length $\le n$.

3.2. An Efficient Dynamic Programming Algorithm

We first illustrate the dynamic programming algorithm with a toy example. As shown in Figure 2, assume that we have generated the first $t-1$ tokens "Kids ... like" and we are given the constraint:

$$\alpha = I(\text{like} \oplus \text{working}) \wedge (I(\text{like} \oplus \text{eating}) \vee I(\text{soccer}));$$

Algorithm 1 Constrained Sampling with GeLaTo

Input: constraint α, maximum text length n, HMM q_1, autoregressive LM q_2, # of HMM latent states h.
for l **from** n **to** 1 **do**
  for x **in** suffixes of keystrings in α, z_l **from** 1 to h **do**
    for ψ **in** subsets of clauses of α **do**
      compute $q_1(x_{l:r}, \psi_{l:n} \mid z_l)$ by the recurrence relation and store values in cache.
    end for
  end for
end for
initialize $x_{1:0}$ = empty string
for t **from** 1 **to** n **do**
  for x_t **in** vocabulary **do**
    compute $q_1(\alpha \mid x_{1:t-1}, x_t)$ by Case 1.
    $p(x_t \mid x_{1:t-1}, \alpha) = q_1(\alpha \mid x_{1:t-1}, x_t) q_2(x_t \mid x_{1:t-1})$
  end for
  sample $x_t \sim p(\cdot \mid x_{1:t-1}, \alpha)$
  update $x_{1:t} := x_{1:t-1} \oplus x_t$
end for
return $x_{1:n}$

here we assume "like", "working", "eating" and "soccer" are single tokens and $\oplus$ denotes string concatenation. To compute $\Pr(x_{1:t-1}, \alpha_{1:n})$, we marginalize out x_t in $\Pr(x_{1:t-1}, x_t, \alpha_{1:n})$; in particular, we sum over three possible cases for the next token x_t: "eating", "working" or neither, and for each case we reduce $\Pr(x_{1:t-1}, x_t, \alpha_{1:n})$ to $\Pr(x_{1:t-1}, x_t, \alpha'_{t:n})$ for some α'; here α' is some CNF formula obtained by removing from the original α the clauses that are already satisfied. Then, we leverage the Markov property of HMMs (see Equation 3) to break down the joint probability $\Pr(x_{1:t-1}, x_t, \alpha'_{t:n})$ into sub-problems.

Before we describe how to compute $\Pr(x_{1:t}, \alpha_{1:n})$, we establish a recurrence relation for computing terms of the form $\Pr(x_{l:r}, \psi_{l:n} \mid z_l)$, where $x_{l:r}$ is either the empty string or a suffix for some keystring in α, ψ is a CNF consisting of a subset of clauses in α and z_l is a latent state for Z_l.

Assumptions & Notations. For simplicity, we make the following *non-overlapping assumption*: for the set of keystrings appearing in α, denoted as $\{w_{ij}\}$, the prefix of w_{ij} cannot be a suffix for w_{pq} for all $ij \neq pq$. We also define the following set of strings:

$$S(x, \alpha) := \{s : \exists x' \text{ a suffix of } x \text{ s.t. } x' \oplus s \text{ lies in } \alpha\},$$

which contains all strings that can be appended to x to form some keystrings in α. For the example in Figure 2, for $x =$ "Kids ... like", $S(x, \alpha)$ is given by {"eating", "working"}. We write $s_{i:j}$ as a shorthand for $X_{i:j} = s$.

Recurrence Relation. $\Pr(x_{l:r}, \psi_{l:n} \mid z_l)$ follows the following recurrence relation:

Case 1. $x_{l:r} \neq \emptyset$; then,

$$\begin{aligned}
&\Pr(x_{l:r}, \alpha_{l:n} \mid z_l) \\
&= \sum_{z_{r+1}} \underline{\Pr(x_{l:r}, z_{r+1} \mid z_l)} \Bigg(\boxed{\Pr(\alpha_{r+1:n} \mid z_{r+1})} \\
&+ \sum_{s \in S(x_{l:r}, \alpha)} \boxed{\Pr(s_{r+1:r+|s|}, (\alpha \setminus x_{l:r} \oplus s)_{r+1:n} \mid z_{r+1})} \\
&- \sum_{s \in S(x_{l:r}, \alpha)} \boxed{\Pr(s_{r+1:r+|s|}, \alpha_{r+1:n} \mid z_{r+1})} \Bigg);
\end{aligned}$$

here $\oplus$ denotes string concatenation and $\alpha \setminus x_{l:r} \oplus s$ represents the CNF obtained by removing the clauses with any keywords appearing in $x_{l:r} \oplus s$.

Case 2. $x_{l:r} = \emptyset$; we reduce the problem to Case 1 by enumerating x_l over the vocabulary:

$$\Pr(\alpha_{l:n} \mid z_l) = \sum_{x_l \in \text{vocabulary}} \boxed{\Pr(x_l, \alpha_{l:n} \mid z_l)};$$

The recurrence relation presented above gives us a dynamic programming algorithm for computing terms of the form $\Pr(x_{l:r}, \psi_{l:n} \mid z_l)$; see appendix for derivations. Note that the boxed terms are the sub-problems and the underlined terms are either HMM parameters or can be pre-computed via the forward algorithm and then cached for later use.

Finally, as discussed at the beginning of this section, we guide autoregressive generation from language models at step t by computing $\Pr(x_{1:t-1}, x_t, \alpha_{1:n})$, where $x_{1:t-1}$ denotes the first $t-1$ tokens that have been generated:

$$\Pr(x_{1:t}, \alpha_{1:n}) = \sum\nolimits_{z_1} \Pr(z_1) \Pr(x_{1:t}, \alpha_{1:n} \mid z_1);$$

here $\Pr(z_1)$ is the initial probability of the HMM and $\Pr(x_{1:t}, \alpha_{1:n} \mid z_1)$ can be computed by the formula in Case 1 (setting $l = 1$), given that all boxed terms are pre-computed by the dynamic programming algorithm.

As an example, Algorithm 1 summarizes how to perform constrained generation with GeLaTo by sampling from the autoregressive distribution $p(x_t \mid x_{1:t-1}, \alpha)$, as defined in Section 2 (unsupervised setting). We can easily adapt Algorithm 1 for other decoding procedures like beam search.

For a rough analysis of the time complexity of Algorithm 1, we treat both the number of latent states h and the vocabulary size as constants; in practice, we can avoid enumerating all tokens in the vocabulary and all latent states of HMM via GPU parallelization (see appendix & code for details). It follows that the time complexity of GeLaTo is $O(2^{|\alpha|} nm)$, where $|\alpha|$ is the number of clauses in α, n is the maximum sequence length and m is the number of different suffixes for all keystrings in α. We show that GeLaTo scales well in practice in Section 4.3.

4. Experiments

In this section, we demonstrate the effectiveness of GeLaTo[2] on challenging benchmarks for constrained generation: CommonGen (Lin et al., 2020), Yelp!Review (Cho et al., 2019) and News (Zhang et al., 2020); in particular, we focus on CommonGen for detailed analysis. For both unsupervised and supervised settings, GeLaTo achieves state-of-the-art performance in terms of various automatic evaluation metrics including BLEU score while guaranteeing 100% constraint satisfaction.

4.1. Dataset & Baselines

CommonGen (Lin et al., 2020) is a benchmark for constrained generation with lexical constraints: the input of each example consists of three to five concepts (keywords) and the goal is to generate a natural sentence using all concepts; in particular, the given keywords can appear in any order or in any form of inflections in the generated sentences. For example, given *"car snow drive"* as concepts, both *"a man drives a car on a snow covered road"* and *"the car drove through the snow"* are considered acceptable. We also evaluate GeLaTo on the Yelp!Review (Cho et al., 2019) and the News (Zhang et al., 2020) datasets. Compared to CommonGen, both Yelp!Review and News share similar formats, except that they require all keywords to be generated in the forms as given (i.e. no inflections allowed) and to follow specific orders.

We compare GeLaTo against constrained generation approaches belonging to different families:

InsNet (Lu et al., 2022a) is a class of insertion-based language models (Susanto et al., 2020) that generate text by repeatedly inserting new tokens into the sequence. InsNet guarantees that the keywords appear in the generated sentence by initializing the token sequence as the keywords, arranged in some order.

NeuroLogic (A*esque) Decoding (Lu et al., 2021; 2022b) are search-based decoding algorithms; they are inference-time algorithms like beam search and do not use any auxiliary models. Leveraging look-ahead heuristics, NeuroLogic A*esque decoding not only optimizes the probability of the generated sentence but also steers the generation towards satisfying the lexical constraints.

NADO (Meng et al., 2022) trains an auxiliary neural model approximating the conditional distribution $\Pr(\alpha|x_{1:t}, x_{t+1})$ to guide constrained generation of the base model. As mentioned in Section 2, NADO needs to re-train the auxiliary model for different types of α (e.g., ten keywords) while GeLaTo does not need re-training.

[2] https://github.com/UCLA-StarAI/GeLaTo

4.2. Approach

Following the experiment setup of Lu et al. (2021) and Meng et al. (2022), we evaluate GeLaTo under both unsupervised and supervised settings, as described in Section 2.

Fine-tuning GPT2-large All baselines, except for InsNet, perform generation with GPT2-large (Radford et al., 2019) as the *base model*. Following prior works (Meng et al., 2022), we use fine-tuned GPT2-large as base models:

1. Unsupervised Setting: we perform *domain adaptation* (DA) by fine-tuning GPT2-large on all gold (reference) sentences of the training split of CommonGen *without* supplementing the keywords. We fine-tune the model for 1 epoch with learning rate = 1e-6.
2. Supervised Setting: following the template proposed in Lin et al. (2020), we fine-tune the GPT2-large model in a *sequence-to-sequence* (seq2seq) manner; in particular we fine-tune the model on sequences of the form *"car snow drive = a car drove through snow"* for 3 epochs with learning rate = 1e-6.

Training HMMs. We use HMMs as an example TPM to enforce lexical constraint in autoregressive generation from GPT2-large. Following Section 4, we sample sequences of length 32 from the fine-tuned GPT2-large models and train HMMs with 4096 hidden states to approximate the base model distributions; we train HMMs with the expectation-maximization (EM) algorithm for 40 epochs, and we re-sample 0.2 million examples for each epoch. The HMM models are trained as probabilistic circuits with the *Juice.jl* framework (Dang et al., 2021) and the training procedure leverages the latent variable distillation technique proposed in Liu et al. (2023); we refer readers to the original papers for more details.

Constraint Formulation. For CommonGen, as described in Section 4.1, the goal is to generate a sentence using the given concepts (keywords) and we encode this lexical constraint as a CNF. For example, given the concepts "catch frisbee snow", the lexical constraint can be represented as:

$$
\begin{aligned}
&[I(\text{catch}) \vee I(\text{caught}) \vee \dots] \\
\wedge&[I(\text{fr} \oplus \text{is} \oplus \text{bee}) \vee I(\text{fr} \oplus \text{is} \oplus \text{bees}) \vee \dots] \\
\wedge&[I(\text{snow}) \vee I(\text{snow} \oplus \text{ing}) \vee I(\text{snow} \oplus \text{ed}) \vee \dots];
\end{aligned}
$$

here each clause encodes the constraint that a keyword has to appear, in any form of its inflections; each literal $I(w)$ indicates the occurrence of a string of tokens w (i.e. keystring), which represents the tokenization of a specific inflection of a keyword and $\oplus$ denotes the concatenation of individual tokens. For the keywords, we use LemmInflect[3] to generate their inflections. We also enforce the constraint that

[3] https://github.com/bjascob/LemmInflect

Method	*Generation Quality*								*Constraint Satisfaction*			
	ROUGE-L		BLEU-4		CIDEr		SPICE		Coverage		Success Rate	
Unsupervised	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*
InsNet (Lu et al., 2022a)	-	-	18.7	-	-	-	-	-	**100.0**	-	**100.0**	-
NeuroLogic (Lu et al., 2021)	-	41.9	-	24.7	-	14.4	-	27.5	-	96.7	-	-
A*esque (Lu et al., 2022b)	-	**44.3**	-	28.6	-	**15.6**	-	29.6	-	97.1	-	-
NADO (Meng et al., 2022)	-	-	26.2	-	-	-	-	-	96.1	-	-	-
GeLaTo	**44.3**	43.8	**30.3**	**29.0**	**15.6**	15.5	**30.2**	**30.3**	**100.0**	**100.0**	**100.0**	**100.0**
Supervised	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*	*dev*	*test*
NeuroLogic (Lu et al., 2021)	-	42.8	-	26.7	-	14.7	-	30.5	-	97.7	-	93.9†
A*esque (Lu et al., 2022b)	-	43.6	-	28.2	-	15.2	-	30.8	-	97.8	-	97.9†
NADO (Meng et al., 2022)	44.4†	-	30.8	-	16.1†	-	32.0†	-	97.1	-	88.8†	-
GeLaTo	**46.2**	**45.9**	**34.0**	**34.1**	**17.2**	**17.5**	**32.2**	**33.5**	**100.0**	**100.0**	**100.0**	**100.0**

Table 1. Performance comparison of different generation methods for *unsupervised* and *supervised* settings on the CommonGen dataset, measured by *generation quality* and *constraint satisfaction*. For hyper-parameter tuning, we conduct cross-validation on a small subset of the training set and report evaluation results for both validation (*dev*) and *t*est set. All methods except for InsNet uses GPT2-large as their base model. Numbers with † are reproduced by ourselves.

each keystring, whenever it appears in the generated text, is followed by either a space, a comma or an ⟨eos⟩ token.

Decoding. $p(x_{t+1} \mid x_{1:t}, \alpha)$ defined in Section 2 (see Eq. 1 and 2) induces the conditional distribution $p(x_{1:n} \mid \alpha) = \prod_t p(x_{t+1} \mid x_{1:t}, \alpha)$. We adopt beam search to greedily search for $x_{1:n}$ that maximizes $p(x_{1:n} \mid \alpha)$; we experiment with different beam sizes: 16, 32, 64 and 128. Finally, we re-rank all beams generated by beam search by their log-likelihood given by the domain-adapted GPT2-large model and select the top beam.

Metrics. We evaluate the quality of generation via human evaluation and some commonly used automatic metrics including ROUGE (Lin & Hovy, 2003), BLEU (Papineni et al., 2002), CIDEr (Vedantam et al., 2015), and SPICE (Anderson et al., 2016). In addition to generation quality, we also measure the constraint satisfaction performance via *coverage*, the average percentage of concepts presented in the generated sentences and *success rate*, the percentage of generated sentences that perfectly satisfy the constraints.

4.3. Results and Analysis

Main evaluation results are presented in Table 1. GeLaTo outperforms all baselines in both unsupervised and supervised settings by a large margin, achieving not only significantly higher BLEU and ROUGE scores but also 100% constraint satisfaction. The unsupervised setting is more challenging given that the base model is never trained with task-specific supervision; despite this, GeLaTo achieves 30.3 BLEU score in the *unsupervised* setting, while NADO (the best performing baseline) obtains 30.8 BLEU score in the *supervised* setting. To provide more insight into GeLaTo, we also conduct the following ablation studies.

Generation Quality vs. Approximation Performance. As discussed in Section 2, GeLaTo assumes that distilled HMMs are good enough approximations for base models; our hypothesis is that the better the HMM approximates the base model, the better the generation quality. With GeLaTo, we generate from different HMM checkpoints from the distillation procedure, and report the average log-likelihoods and BLEU scores (without re-ranking the beams). As shown in Figure 3, as the training proceeds, both log-likelihood and BLEU score improves, exhibiting a clear positive correlation. This finding motivates the development of better tractable probabilistic models for language modeling.

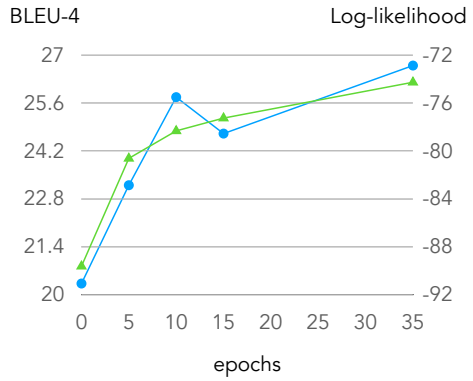

Figure 3. HMM log-likelihoods on data sampled from GPT-2 large (triangles) and the corresponding BLEU scores (circles) w.r.t. # of training epochs. As the HMM model approximates GPT2-large better, the generation quality also improves.

Robustness of Hyperparameter w. As described in Section 2, for the supervised setting, the formulation of GeLaTo involves a hyperparameter $0 \leq w \leq 1$ that decides how much the TPM or the base model contributes to generation. For our experiments, w is set to 0.3 based on cross-validation results on the training set. Figure 4 shows the BLEU score (after re-ranking) on the validation set of CommonGen given different values of w. The performance of GeLaTo is very

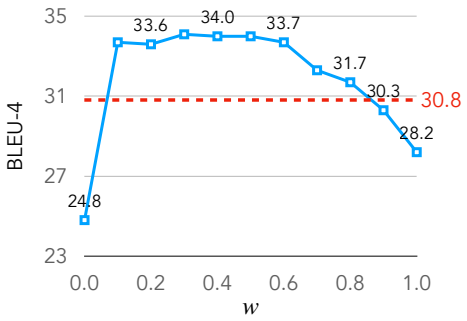

Figure 4. BLEU score on CommonGen (dev) for different values of w. GeLaTo achieves SoTA performance for $0.1 \leq w \leq 0.8$.

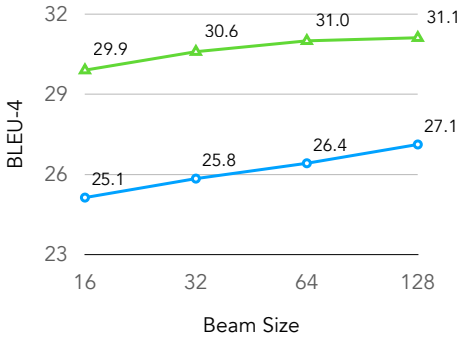

Figure 5. BLEU score (y-axis) obtained by GeLaTo on CommonGen (dev), with various beam-sizes (x-axis), for both unsupervised (circles) and supervised (triangles) settings.

robust with respect to different choices of w, achieving SoTA BLEU scores for $0.1 \leq w \leq 0.8$.

Effect of Beam Size. GeLaTo uses beam search for generation and we study how its performance is affected by the choice of beam size. Figure 5 shows that for both unsupervised and supervised settings, the performance of GeLaTo improves monotonically as the beam size increases.

Run-time Comparison. We conduct an empirical evaluation of the run-time (in seconds) of GeLaTo on CommonGen, in comparison to NeuroLogic A*esque and vanilla GPT2-large; all methods are evaluated on a single NVIDIA A100 GPU with 40 GB memory; the run-time is measured on 100 randomly sampled examples for each # of concepts.

GeLaTo achieves its best performance with beam-size=128; yet we also report the run-time for beam-size=16, where it achieves performance better than all baselines. For the unsupervised setting, GeLaTo is much faster than NeuroLogic A*esque, which suffers from an unconstrained search space. For the supervised setting, GeLaTo is slower than A*esque but the run-time for beam-size = 16 is still comparable.

# of concepts	3	4	5
Unsupervised			
A*esque	472.9	542.5	613.9
GeLaTo (16)	13.5 ± 4.4	21.9 ± 5.37	39.3 ± 6.3
GeLaTo (128)	69.8 ± 32.3	97.9 ± 39.5	143.0 ± 44.4
Supervised			
A*esque	8.5	9.6	11.4
GPT2 (16)	5.8 ± 1.1	13.0 ± 1.6	29.3 ± 3.2
GPT2 (128)	9.4 ± 1.8	21.1 ± 11.9	33.7 ± 3.5
GeLaTo (16)	11.1 ± 2.8	22.0 ± 5.0	41.6 ± 5.6
GeLaTo (128)	49.8 ± 20.8	88.7 ± 30.5	127.6 ± 30.4

Table 2. Time of generating one example (seconds) on CommonGen (dev). Results for NeuroLogic A*esque, finetuned GPT2-large and GeLaTo are reported; beam-sizes are shown in parentheses.

Human Evaluation. We conduct human evaluation for sentences generated on CommonGen (dev), following the setup of prior works (Lu et al., 2022b; Meng et al., 2022). Specifically, we mix sentences generated by different methods, and each sentence is presented to one human annotator to be evaluated on four aspects: concepts, plausibility, quality, and overall rating. The results are shown in Table 3. To test statistical significance, we conduct the Wilcoxon signed rank two-sided test with p-value < 0.05 and GeLaTo performs best in all metrics compared to prior SoTA. We refer readers to Appendix B for details of human evaluation.

Method	Concepts	Plausibility	Quality	Overall
GPT2	2.47	**2.52**	2.65	2.28
NADO	**2.71**	**2.54**	**2.73**	2.54
GeLaTo	**2.73**	**2.52**	**2.70**	**2.60**

Table 3. Human evaluation results on CommonGen for fine-tuned GPT2-Large, NADO and GeLaTo, all under the supervised setting.

Method \Dataset	Yelp!Review	News
InsNet	5.8	5.0
NADO	6.0	4.5
GeLaTo	**6.6**	**5.4**

Table 4. BLEU-4 scores for Yelp!Review and News datasets; for InsNet and NADO we present the best results of all settings while the results of GeLaTo are obtained under the unsupervised setting.

Fixing Order of Keywords. Following prior works (Meng et al., 2022; Lu et al., 2022a), we evaluate GeLaTo on Yelp!Review and News datasets. They are more challenging in that they require keywords to appear in specific orders; besides, the average sequence lengths for both datasets are approximately 64 tokens, twice of that of CommonGen. With a minor modification to Algorithm 1, GeLaTo is easily adapted to generate text with ordered keywords. For both datasets, the training examples do not provide keywords thus there is no immediate way to finetune the base models in a supervised way. Yet, as shown in Table 4.3, the unsupervised GeLaTo alone achieves SoTA BLEU scores.

5. Related Works

5.1. Tractable Probabilistic Models

Tractable probabilistic models support efficient probabilistic inference (e.g., marginal probability), thus they have been widely used in inference-demanding tasks, including enforcing algorithmic fairness (Choi et al., 2020a; 2021), and making predictions under missing data (Khosravi et al., 2019; Correia et al., 2020; Li et al., 2021; Dang et al., 2022b).

Probabilistic circuits (PCs) is a unified framework for a large family of tractable probabilistic models including hidden Markov models (Rabiner & Juang, 1986), bounded tree-width graphical models (Meila & Jordan, 2000) and sum-product networks (SPNs) (Poon & Domingos, 2011). Recent progress in learning probabilistic circuits for generative modeling (Dang et al., 2022c; Liu et al., 2023) and their efficient implementation (Molina et al., 2019; Peharz et al., 2020; Dang et al., 2021) have been pushing the limits of PC's expressive power.

5.2. Enforcing Constraints in Neural Networks

The capacity of deep generative models is continuously increasing, while their probabilistic and logic querying ability is restricted. A variety of methods have been developed. Boyd et al. (2022) introduce a general inference typology on autoregressive sequence models that can develop query estimation methods based on beam search and importance sampling. Ahmed et al. (2022) use PCs as a replacement for the SoftMax layer in neural networks such that their outputs are guaranteed to satisfy the constraint.

5.3. Controllable Autoregressive Language Generation

One line of research on constrained text generation focuses on modifying the decoding algorithm to inject constraints into the beam search process, such as constrained beam search (Post & Vilar, 2018), NeuroLogic Decoding (Lu et al., 2021) and A*esque NeuroLogic Decoding (Lu et al., 2022b). Though they can be easily applied to various language models without training, these search-based methods can be inefficient as they suffer from large search spaces. Recent works like NADO (Meng et al., 2022) and FUDGE (Yang & Klein, 2021) train auxiliary neural models to provide token-level guidance for autoregressive generation. Another family of approaches that enforce keyword-type constraints are insertion-based language models (Lu et al., 2022a; Susanto et al., 2020), where the initial sequences only consist of the desired keywords and the transition phrases are repeatedly inserted to complete the sentences.

6. Conclusion

In this paper, we propose GeLaTo, where we use tractable probabilistic models (TPMs) to impose complex lexical constraints (denoted α) in autoregressive language generation from large language models. Specifically, we provide token-level guidance to autoregressive generation by computing $\Pr_{\text{TPM}}(x_{t+1} \mid x_{1:t}, \alpha)$. With hidden Markov model as a running example, we (1) present an efficient dynamic programming algorithm for conditioning HMMs on complex lexical constraints and (2) demonstrate the effectiveness of GeLaTo on various constrained generation benchmarks; GeLaTo achieves state-of-the-art generation quality (i.e. BLEU-4 scores) while guaranteeing 100% constraint satisfaction. This work opens up new avenues for constrained language generation and motivates for the development of more expressive tractable probabilistic models.

Acknowledgements

This work was funded in part by the DARPA Perceptually-enabled Task Guidance (PTG) Program under contract number HR00112220005, NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129, an SRA from Meta, a research gift from Amazon Alexa AI, and a gift from RelationalAI. GVdB discloses a financial interest in RelationalAI.

References

Ahmed, K., Teso, S., Chang, K.-W., Van den Broeck, G., and Vergari, A. Semantic probabilistic layers for neuro-symbolic learning. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

Anderson, P., Fernando, B., Johnson, M., and Gould, S. Spice: Semantic propositional image caption evaluation. In *European Conference on Computer Vision (ECCV)*. Springer, 2016.

Bahdanau, D., Cho, K. H., and Bengio, Y. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

Bekker, J., Davis, J., Choi, A., Darwiche, A., and Van den Broeck, G. Tractable learning for complex probability queries. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.

Boyd, A. J., Showalter, S., Mandt, S., and Smyth, P. Predictive querying for autoregressive neural sequence models. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

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

Advances in Neural Information Processing Systems 33 (NeurIPS), 2020.

Chiu, J. and Rush, A. M. Scaling hidden markov language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Cho, W. S., Zhang, P., Zhang, Y., Li, X., Galley, M., Brockett, C., Wang, M., and Gao, J. Towards coherent and cohesive long-form text generation. In *Proceedings of the First Workshop on Narrative Understanding*, pp. 1–11, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-2401. URL https://aclanthology.org/W19-2401.

Choi, A., Van den Broeck, G., and Darwiche, A. Probability distributions over structured spaces. In *Proceedings of the AAAI Spring Symposium on KRR*, 2015.

Choi, Y., Farnadi, G., Babaki, B., and Van den Broeck, G. Learning fair naive bayes classifiers by discovering and eliminating discrimination patterns. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2020a.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020b.

Choi, Y., Dang, M., and Van den Broeck, G. Group fairness by probabilistic modeling with latent fair decisions. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, 2021.

Correia, A., Peharz, R., and de Campos, C. P. Joints in random forests. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

Dang, M., Khosravi, P., Liang, Y., Vergari, A., and Van den Broeck, G. Juice: A julia package for logic and probabilistic circuits. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (Demo Track)*, 2021.

Dang, M., Liu, A., and Van den Broeck, G. Sparse probabilistic circuits via pruning and growing. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022a.

Dang, M., Liu, A., Wei, X., Sankararaman, S., and Van den Broeck, G. Tractable and expressive generative models of genetic variation data. In *Proceedings of the International Conference on Research in Computational Molecular Biology (RECOMB)*, 2022b. doi: https://doi.org/10.1007/978-3-031-04749-7_26.

Dang, M., Vergari, A., and Van den Broeck, G. Strudel: A fast and accurate learner of structured-decomposable probabilistic circuits. *International Journal of Approximate Reasoning*, 2022c. ISSN 0888-613X.

Grover, A. and Ermon, S. Boosted generative models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 2002.

Khosravi, P., Choi, Y., Liang, Y., Vergari, A., and Van den Broeck, G. On tractable computation of expected predictions. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

Kisa, D., Van den Broeck, G., Choi, A., and Darwiche, A. Probabilistic sentential decision diagrams. In *Proceedings of the 14th International Conference on the Principles of Knowledge Representation and Reasoning (KR)*, 2014.

Kulesza, A. and Taskar, B. Determinantal point processes for machine learning. *Foundations and Trends® in Machine Learning*, 2012.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Li, W., Zeng, Z., Vergari, A., and Van den Broeck, G. Tractable computation of expected kernels. In *Proceedings of the 37th Conference on Uncertainty in Aritifical Intelligence (UAI)*, 2021.

Lin, B. Y., Zhou, W., Shen, M., Zhou, P., Bhagavatula, C., Choi, Y., and Ren, X. Commongen: A constrained text generation challenge for generative commonsense reasoning. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020.

Lin, C.-Y. and Hovy, E. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2003.

Liu, A., Zhang, H., and Van den Broeck, G. Scaling up probabilistic circuits by latent variable distillation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

Liu, F., Flanigan, J., Thomson, S., Sadeh, N., and Smith, N. A. Toward abstractive summarization using semantic representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2015.

Lu, S., Meng, T., and Peng, N. Insnet: An efficient, flexible, and performant insertion-based text generation model. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022a.

Lu, X., West, P., Zellers, R., Le Bras, R., Bhagavatula, C., and Choi, Y. Neurologic decoding:(un) supervised neural text generation with predicate logic constraints. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2021.

Lu, X., Welleck, S., West, P., Jiang, L., Kasai, J., Khashabi, D., Le Bras, R., Qin, L., Yu, Y., Zellers, R., Smith, N. A., and Choi, Y. NeuroLogic a*esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2022b.

Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

Meila, M. and Jordan, M. I. Learning with mixtures of trees. *Journal of Machine Learning Research*, (Oct), 2000.

Meng, T., Lu, S., Peng, N., and Chang, K.-W. Controllable text generation with neurally-decomposed oracle. In *Advances in Neural Information Processing Systems 35 (NeurIPS)*, 2022.

Molina, A., Vergari, A., Stelzner, K., Peharz, R., Subramani, P., Di Mauro, N., Poupart, P., and Kersting, K. Spflow: An easy and extensible library for deep probabilistic learning using sum-product networks. *arXiv preprint arXiv:1901.03704*, 2019.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics (ACL)*, 2002.

Pearl, J. Bayesian netwcrks: A model cf self-activated memory for evidential reasoning. In *Proceedings of the 7th conference of the Cognitive Science Society, University of California, Irvine, CA, USA*, 1985.

Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Van den Broeck, G., Kersting, K., and Ghahramani, Z. Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

Poon, H. and Domingos, P. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE, 2011.

Post, M. and Vilar, D. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL) (Long Papers)*, 2018.

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

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 2019.

Roth, D. On the hardness of approximate reasoning. *Artificial Intelligence*, 1996.

Satopää, V. A., Baron, J., Foster, D. P., Mellers, B. A., Tetlock, P. E., and Ungar, L. H. Combining multiple probability predictions using a simple logit model. *International Journal of Forecasting*, 2014.

Susanto, R. H., Chollampatt, S., and Tan, L. Lexically constrained neural machine translation with levenshtein transformer. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

Tian, Y. and Peng, N. Zero-shot sonnet generation with discourse-level planning and aesthetics features. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2022.

Vedantam, R., Lawrence Zitnick, C., and Parikh, D. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

Xu, J. and Durrett, G. Neural extractive text summarization with syntactic compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

Yang, K. and Klein, D. Fudge: Controlled text generation with future discriminators. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, 2021.

Yao, L., Peng, N., Weischedel, R., Knight, K., Zhao, D., and Yan, R. Plan-and-write: Towards better automatic

storytelling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Zhang, H., Juba, B., and Van den Broeck, G. Probabilistic generating circuits. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Zhang, H., Li, L. H., Meng, T., Chang, K.-W., and Van den Broeck, G. On the paradox of learning to reason from data. *arXiv preprint arXiv:2205.11502*, 2022.

Zhang, Y., Wang, G., Li, C., Gan, Z., Brockett, C., and Dolan, B. POINTER: Constrained progressive text generation via insertion-based generative pre-training. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8649–8670, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.698. URL https://aclanthology.org/2020.emnlp-main.698.

A. Recurrence Relation Analysis

We establish the recurrence relation for computing $\Pr(x_{l:r}, \alpha_{l:n} \mid z_l)$; there are two possible cases:

Case 1. $x_{l:r} \neq \emptyset$; in this case, we can append $s \in S(x_{l:r}, \alpha)$ to $x_{l:r}$ to reduce the number of clauses in α; abusing notation, we write $s_{i:j}$ as a shorthand for $X_{i:j} = s$:

$$\begin{aligned}
&\Pr(\alpha_{l:n} \mid z_{r+1}, x_{l:r}, z_l) \\
&\quad = \Pr(X_{r+1:r+|s|} \neq s \; \forall s \in S(x_{l:r}, \alpha), \alpha_{l:n} \mid z_{r+1}, x_{l:r}, z_l) + \sum_{s \in S(x_{l:r}, \alpha)} \Pr(s_{r+1:r+|s|}, \alpha_{l:n} \mid z_{r+1}, x_{l:r}, z_l) \\
&\quad = \Pr(X_{r+1:r+|s|} \neq s \; \forall s \in S(x, \alpha), \alpha_{r+1:n} \mid z_{r+1}) + \sum_{s \in S(x_{l:r}, \alpha)} \Pr(s_{r+1:r+|s|}, (\alpha \setminus x_{l:r} \oplus s)_{r+1:n} \mid z_{r+1});
\end{aligned}$$

here $\oplus$ denotes string concatenation and $\alpha \setminus x_{l:r} \oplus s$ represents the CNF obtained by removing the clauses with any keywords appearing in $x_{l:r} \oplus s$. In particular, the second step in the derivation above follows from the non-overlapping assumption and the independence property of HMMs; then, by expanding the second term, we have:

$$\begin{aligned}
&\Pr(\alpha_{l:n} \mid z_{r+1}, x_{l:r}, z_l) \\
&\quad = \boxed{\Pr(\alpha_{r+1:n} \mid z_{r+1})} \\
&\qquad + \sum_{s \in S(x_{l:r}, \alpha)} \boxed{\Pr(s_{r+1:r+|s|}, (\alpha \setminus x_{l:r} \oplus s)_{r+1:n} \mid z_{r+1})} - \sum_{s \in S(x_{l:r}, \alpha)} \boxed{\Pr(s_{r+1:r+|s|}, \alpha_{r+1:n} \mid z_{r+1})};
\end{aligned}$$

finally, by summing over all hidden states z_{r+1}:

$$\Pr(x_{l:r}, \alpha_{l:n} \mid z_l) = \sum_{z_{r+1}} \underline{\Pr(x_{l:r}, z_{r+1} \mid z_l)} \Pr(\alpha_{l:n} \mid z_{r+1}, x_{l:r}, z_l)$$

Case 2. When $x = \emptyset$, we can reduce the computation of $\Pr(\alpha_{l:n} \mid z_l)$ to Case 1. by summing over all possible tokens at position l:

$$\Pr(\alpha_{l:n} \mid z_l) = \sum_{x_l \in \text{vocabulary}} \Pr(x_l, \alpha_{l:n} \mid z_l) = \sum_{S(x_l, \alpha) \neq \emptyset} \boxed{\Pr(x_l, \alpha_{l:n} \mid z_l)} + \sum_{S(x_l, \alpha) = \emptyset} \boxed{\Pr(x_l, \alpha_{l:n} \mid z_l)}$$

In practice, the vocabulary size is usually large (e.g., 50k), and most tokens lie in $\{x_l : S(x_l, \alpha) = \emptyset\}$. To avoid repetitive computation, we re-write $\sum_{S(x_l, \alpha) = \emptyset} \Pr(x_l, \alpha_{l:n} \mid z_l)$:

$$\begin{aligned}
&\textstyle\sum_{S(x_l, \alpha) = \emptyset} \Pr(x_l, \alpha_{l:n} \mid z_l) \\
&\quad = \textstyle\sum_{S(x_l, \alpha) = \emptyset} \sum_{z_{l+1}} \Pr(x_l, \alpha_{l:n}, z_{l+1} \mid z_l) \\
&\quad = \left(\textstyle\sum_{S(x_l, \alpha) = \emptyset} \underline{\Pr(x_l \mid z_l)} \right) \cdot \left(\textstyle\sum_{z_{l+1}} \underline{\Pr(z_{l+1} \mid z_l)} \boxed{\Pr(\alpha_{l+1:n} \mid z_{l+1})} \right)
\end{aligned}$$

where $\sum_{z_{l+1}} \Pr(z_{l+1} \mid z_l) \Pr(\alpha_{l+1:n} \mid z_{l+1})$ does not depend on x_l and the summation $\sum_{S(x_l, \alpha) = \emptyset} \Pr(x_l \mid z_l)$ can be efficiently computed with CUDA parallelization without enumerating over all tokens.

B. Human Evaluation Setup

The following screenshot shows the human evaluation setup for CommonGen. We consider *Yes* as 3 points, *Somewhat* as 2 points and *No* as 1 point.

Read the given concepts and sentence below and indicate how much you agree with the statements. (Yes, Somewhat, No)

Concepts: ${concepts}

Sentence: ${sentence}

- (1) **Sentence Quality**: Is the sentence ***well-formed***?

  ○ **Yes**: The sentence is **well-formed** and **fluent**.
  ○ **Somewhat**: The sentence is **understandable** but a bit awkward.
  ○ **No**: The sentence is **neither** well-formed or fluent.

- (2) **Plausibility**: Does the sentence describe a plausible scenario?

  ○ **Yes**: The sentence describes a **realistic** or **plausible** scenario.
  ○ **Somewhat**: The sentence describes an **acceptable** scenario but a bit awkward.
  ○ **No**: The sentence describes a **nonsensical** scenario.

- (3) **Concepts**: Does the sentence include the given concepts meaningfully?

  Example: if "run" is a given concept, sentence should include word "ran", "running" or other variant forms of "run". Synonyms like "jog" are **not** allowed.

  ○ **Yes**: The sentence **meaningfully** includes **all** of the concepts.
  ○ **Somewhat**: The sentence meaningfully includes some, but not all of the concepts. Or, the sentence includes all concepts but some of them are not meaningful or properly incorporated.
  ○ **No**: The sentence **does not** include concepts in a meaningful way.

- (4) **Overall**: Considering your answers to 1), 2) and 3), does the sentence meaningfully combine all of the concepts into a well-formed and plausible scenario?

  ○ **Yes**: The sentence is reasonably well-formed/understandable, and meaningfully combines **all** the concepts into a plausible scenario.
  ○ **Somewhat**: The sentence looks okay in terms of above questions.
  ○ **No**: The sentence is not well-formed/understandable, or fails to properly combine **all** the concepts into a plausible scenario.

Submit