# Syntactic Control of Language Models by Posterior Inference

**Vicky Xefteri** **Tim Vieira** **Ryan Cotterell** **Afra Amini**
{vxefteri, ryan.cotterell, afra.amini}@ethz.ch tim.f.vieira@gmail.com
ETH zürich

## Abstract

Controlling the syntactic structure of text generated by language models is valuable for applications requiring clarity, stylistic consistency, or interpretability, yet it remains a challenging task. In this paper, we argue that sampling algorithms based on the posterior inference can effectively enforce a target constituency structure during generation. Our approach combines sequential Monte Carlo, which estimates the posterior distribution by sampling from a proposal distribution, with a syntactic tagger that ensures that each generated token aligns with the desired syntactic structure. Our experiments with `GPT2` and `Llama3-8B` models show that with an appropriate proposal distribution, we can improve syntactic accuracy, increasing the F1 score from 12.31 (`GPT2-large`) and 35.33 (`Llama3-8B`) to about 93 in both cases without compromising the language model's fluency. These results underscore both the complexity of syntactic control and the effectiveness of sampling algorithms, offering a promising approach for applications where precise control over syntax is essential.

github.com/rycolab/syntactic-control

**User**:
- `(S (NP (EX ?)) (VP (VBZ ?) (ADVP (RB ?)) (NP (DT ?) (NN ?))))`

**System**:
- `(S (NP (EX There)) (VP (VBZ is) (ADVP (RB always)) (NP (DT a) (NN chance))))` $[\varphi = 0.99, p = 2.95e^{-13}]$
- `(S (NP (EX There)) (VP (VBZ is) (ADVP (RB never)) (NP (DT a) (NN reason))))` $[\varphi = 0.99, p = 8.66e^{-14}]$

⋮

**`GPT4` baseline**:
- `(S (NP (EX There)) (VP (VBZ is) (ADVP (RB clearly)) (NP (DT a)` ~~`(JJ big)`~~ `(NN problem))))` $[\varphi = 0, p = 1.90e^{-14}]$

Figure 1: Example user–system interaction. The user specifies a target syntactic structure as a Penn Treebank syntax tree with `?` in the place of words. The system probabilistically fills in the missing words, such that the probability of the generated sentence is high under the LM ($p$) and the likelihood of the targeted syntactic structure is high under the syntactic analyzer ($\varphi$). `GPT4` fails to generate a valid string, as it generates an additional adjective `(JJ big)`, making it impossible for the sequence of words to have the target syntax (i.e., $\varphi = 0$).

## 1 Introduction

Syntactic control of generated text is crucial for many domain-specific applications of language models, where structural constraints, such as formality, grammatical correctness, or adherence to a given template, can significantly affect usability and readability. Despite recent advances, achieving fine-grained syntactic control remains a challenge for large-scale language models. More broadly, this challenge falls under the area of *controlled* generation, which studies how to guide language models to produce text with desired properties.

Controlling the syntactic structure of sentences is crucial in some domains. For instance, maintaining stylistic consistency is important in fields such as legal, technical, and formal writing, as well as in creative writing, where sentence structure significantly contributes to tone and clarity. In educational settings, controlling syntactic complexity and ambiguity enables intelligent grammar tutoring in both native and foreign languages (Renduchintala et al., 2016). Syntactic control is also valuable for generating psycholinguistic stimuli, allowing studies on how syntax impacts cognitive load, memory, or reading time (Britton et al., 1982; Vos et al., 2001). More generally, it can support the generation of text that is easier to comprehend, for example, by avoiding deeply nested structures or long, ambiguous sentences (Kauchak, 2013; Xu et al., 2015), and reduce semantic ambiguity (e.g., prepositional attachment ambiguities), which can leave readers confused or increase cognitive processing effort. Finally, from an interpretability perspective, syntactic control provides a framework for analyzing how syntax affects the behavior of language models (Linzen et al., 2016; Futrell et al., 2019).

There are many techniques in the literature that perform controlled generation. One line of work in controlled generation[1] has focused on developing inference-time algorithms that control for qualities of generated text, such as topic, sentiment, and toxicity. Perhaps, the most relevant work focuses on generating code subject to the syntactic constraints of a *programming language*.[2] Our work, in contrast, considers controlled generation under the constraints of *natural language* syntax. Our setting is challenging because natural language does not have simple context-free rules and, thus, cannot be easily checked for violations during generation (e.g., to rule out poor choices from being sampled). Moreover, unlike programming languages, natural language syntax is much richer, inherently ambiguous, and harder to analyze, making it an interesting new challenge for controlled generation methods.

An alternative to controlled generation is prompting instruction-tuned models. Recent studies (Sun et al., 2023; Ashok and Poczos, 2024) have shown the promise of instruction-tuned LMs for following specific control targets, such as generating text on a particular topic or with a positive sentiment. A previously noted limitation of prompting instruction-tuned models is their difficulty with syntactic control, where prompting fails to guide models to produce text with the desired syntactic structure (Sun et al., 2023; Ashok and Poczos, 2024).

It is perhaps unsurprising that prompting alone is insufficient for reliably controlling syntax. Enforcing a global structure upon the generated string is inherently challenging. To improve syntactically controlled generation at inference time, we propose a sampling method that approximates the posterior distribution over strings generated by a language model under a target syntactic structure. Our approach is based on sequential Monte Carlo, an algorithm that estimates the posterior by drawing samples from a proposal distribution and weighting them by the likelihood that the given string follows a specific attribute, in our case, a syntax tree. In this paper, we use parsers-as-taggers[3] to further guide the generation towards samples with higher likelihood. We argue that taggers can provide effective guidance since they, by design, factorize the syntactic structure and align it with each word in the sequence. Specifically, we use Tetratagger (Kitaev and Klein, 2020), which produces a constituency analysis through a clever linear encoding of the constituency tree that assigns a pair of tags to each word. While the original is trained on a masked language model, we also train Tetratagger with an autoregressive language model as its backbone to guide sampling.

Our experiments with `GPT4` (OpenAI, 2024) align with previous findings (Sun et al., 2023), showing that the `GPT4` model has difficulty generating sentences with the desired constituency trees in zero- or few-shot settings. We then apply our method to `GPT2-large` (Radford et al., 2019), as well as to `Llama3-8B` models (Llama Team, 2024). Sentences generated using our method adhere much more closely to the target syntax than those produced without it. For instance, with a well-chosen proposal distribution, our method improves the F1 score of `GPT2-large` generations from 12.31 to 93.69 without degrading the model's fluency. Moreover, our method can be applied to instruction-tuned models like `Llama3-8B`, achieving a similar F1 score with `GPT2-large`. These results demonstrate that controlled generation by posterior inference can make smaller models competitive with larger ones, like `GPT4`.

## 2 Controlled Generation by Posterior Inference

A **language model** (**LM**) $p$ is a probability distribution over strings $\boldsymbol{y} \in \Sigma^*$, where $\Sigma$ is the set of tokens in the vocabulary. Most state-of-the-art language models are factored into per-token conditional distributions. The probability of a string $\boldsymbol{y}$ under a language model is given by

$$p(\boldsymbol{y}) \overset{\text{def}}{=} p(\text{EOS} \mid \boldsymbol{y}) \prod_{n=1}^{|\boldsymbol{y}|} p(y_n \mid \boldsymbol{y}_{<n}), \tag{1}$$

where $\boldsymbol{y}_{<n} \overset{\text{def}}{=} y_1 \cdots y_{n-1}$, and $\text{EOS} \notin \Sigma$ is a distinguished, end-of-string token.

Language models contain a vast amount of prior knowledge about what constitutes well-formed and fluent natural language. However, they have limited inherent understanding of external constraints or specific control targets. In the context of this work, the language model serves as a **prior** probability $p(\boldsymbol{y})$ that a string $\boldsymbol{y}$ is desirable.[4] In order to *con-*

[1]E.g., Krause et al. (2021); Liu et al. (2021); Schick et al. (2021); Lew et al. (2023); Zhao et al. (2024).

[2]E.g., Scholak et al. (2021); Poesia et al. (2022); Ugare et al. (2024); Loula et al. (2025).

[3]E.g., Gómez-Rodríguez and Vilares (2018), Vacareanu et al. (2020), and Amini and Cotterell (2022).

[4]One may freely modify this prior, e.g., applying top-$p$ filtering, annealing, or including instructions, to account for preferences that are independent of the target syntax.

*trol* generation, we define an additional factor, the **likelihood** $\varphi(\mathtt{t} \mid \boldsymbol{y})$ that $\boldsymbol{y}$ possesses the desired attribute $\mathtt{t}$.[5] Together, the prior and the likelihood define a **posterior** distribution $P(\boldsymbol{y} \mid \mathtt{t})$ that accounts for the inherent uncertainties in each distribution:

$$\overbrace{P(\boldsymbol{y} \mid \mathtt{t})}^{\text{posterior}} = \frac{\overbrace{p(\boldsymbol{y})}^{\text{prior}} \, \overbrace{\varphi(\mathtt{t} \mid \boldsymbol{y})}^{\text{likelihood}}}{\underbrace{Z(\mathtt{t})}_{\text{evidence}}} \tag{2a}$$

$$Z(\mathtt{t}) \overset{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} p(\boldsymbol{y}) \, \varphi(\mathtt{t} \mid \boldsymbol{y}) \tag{2b}$$

Note that $Z(\mathtt{t})$ is the probability that any string generated by $p$ has the attribute $\mathtt{t}$. Unfortunately, the exact computation of $Z(\mathtt{t})$ and exact sampling from $P(\boldsymbol{y} \mid \mathtt{t})$ are generally intractable. However, we will present principled, practical approximations in §§2.1 and 2.2.

**Likelihood.** Note that $\varphi(\mathtt{t} \mid \boldsymbol{y})$ is defined by a neural model that has been trained to approximate the true association between the target attribute $\mathtt{t}$and strings $\boldsymbol{y}$. In the case of categorical labels,[6] $\varphi(\mathtt{t} \mid \boldsymbol{y})$ may be a probabilistic classifier trained to predict a label $\mathtt{t}$for each string $\boldsymbol{y}$ (e.g., $\mathtt{t}$could be a positive or negative sentiment judgment, and $\varphi(\mathtt{t} \mid \boldsymbol{y})$ can be learned from a sentiment classification corpus). However, in this paper, $\mathtt{t}$ is a user-specified syntax tree where the internal nodes of a syntax tree are provided, but the words are left for the LM to provide (see Fig. 1 for an illustration). Thus, $\varphi(\mathtt{t} \mid \boldsymbol{y})$ is a rich probabilistic model that assigns conditional probabilities to syntactic analyses $\mathtt{t}$ for each string $\boldsymbol{y}$;[7] see Fig. 1 for an illustration. We describe our syntactic likelihood model in §3. In general, the likelihood doesn't necessarily need to be a conditional probability distribution. We can view our abstract problem as

$$\overbrace{P(\boldsymbol{y})}^{\text{posterior}} = \frac{\overbrace{p(\boldsymbol{y})}^{\text{prior}} \, \overbrace{\varphi(\boldsymbol{y})}^{\text{potential}}}{\underbrace{Z}_{\text{evidence}}} \tag{3a}$$

$$Z \overset{\text{def}}{=} \sum_{\boldsymbol{y} \in \Sigma^*} p(\boldsymbol{y}) \, \varphi(\boldsymbol{y}), \tag{3b}$$

where $\varphi(\boldsymbol{y}) \geq 0$ (for all $\boldsymbol{y}$) is often called a **potential** function, rather than a likelihood function.

## 2.1 Importance Sampling

**Importance sampling (IS)** offers a practical approximation to the posterior distribution $P(\boldsymbol{y})$.[8] Given $\varphi$ and a sample size $M$, importance sampling works as follows:

1. Sample $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(M)} \overset{\text{i.i.d.}}{\sim} Q$ where $Q$ is a **proposal distribution**, which is another language model that should approximate the posterior.[9] We refer to each of these samples as a **particle**.
2. Evaluate the potential $\varphi(\boldsymbol{y}^{(m)})$, and compute the particle **weight** $w(\boldsymbol{y}) \overset{\text{def}}{=} \frac{p(\boldsymbol{y})}{Q(\boldsymbol{y})} \varphi(\boldsymbol{y})$. Let $w^{(m)} = w(\boldsymbol{y}^{(m)})$.
3. Compute the posterior approximation:

$$\widehat{P}(\boldsymbol{y}) \overset{\text{def}}{=} \frac{1}{M \, \widehat{Z}} \sum_{m=1}^{M} w^{(m)} \mathbb{1}\{\boldsymbol{y} = \boldsymbol{y}^{(m)}\} \tag{4a}$$

$$\widehat{Z} \overset{\text{def}}{=} \frac{1}{M} \sum_{m=1}^{M} w^{(m)} \tag{4b}$$

Now, to draw (approximate) samples, we draw strings $\boldsymbol{y}$ from the posterior approximation $\widehat{P}$, which is efficient as there are at most $M$ strings with nonzero probability. Importance sampling comes with the following two guarantees:

- $\widehat{Z}$ is an unbiased estimate of $Z$
- $\widehat{P}(\boldsymbol{y})$ is consistent estimate of $P(\boldsymbol{y})$, i.e., it converges as $M \to \infty$.

***Sequential* importance sampling (SIS).** Sequential importance sampling (e.g., Doucet et al., 2001) is an implementation of importance sampling tailored for sequences, where samples are drawn from the conditional proposal distribution and weights are computed incrementally at each step. The specific details are given in Alg. 1, but it is equivalent to importance sampling with the addition of the boolean $\alpha$ that tracks which particles are still incomplete (not yet EOS-terminated).

## 2.2 Sequential Monte Carlo

The drawback of importance sampling is that it often allocates computation (i.e., sampling budget)

[5]We use $\varphi$ to denote the likelihood to prevent confusion with the conditional distribution over tokens used in Eq. (1).

[6]E.g., sentiment, topic, and toxicity (Yang and Klein, 2021; Liu et al., 2021; Amini et al., 2023, *inter alia*).

[7]Note that our approach differs from other syntax-constrained generation methods (e.g., Shin et al., 2021; Scholak et al., 2021; Poesia et al., 2022; Ugare et al., 2024) because our control signal $\varphi(\mathtt{t} \mid \boldsymbol{y})$ (a) is probabilistic, (b) is a complex model rather than a simplistic context-free grammar.

[8]We refer the reader to Chatterjee and Diaconis (2017) for a thorough analysis of importance sampling.

[9]Ideally, we want a proposal distribution $Q \approx P$. However, for correctness, it is sufficient that whenever $Q(\boldsymbol{y}) = 0$, we also have $p(\boldsymbol{y}) \, \varphi(\boldsymbol{y}) = 0$. If this condition is not satisfied, then any string $\boldsymbol{y}$ with $p(\boldsymbol{y}) \, \varphi(\boldsymbol{y}) \neq 0$ but $Q(\boldsymbol{y}) = 0$, will never be sampled, leading to a false zero in the posterior approximation that cannot be resolved by taking more samples.

**Algorithm 1** Sequential importance sampling

```
1:  procedure SIS($p, Q, \varphi, M$)
2:    for $m = 1 \dots M$ :                              ▷ $M$ number of particles
3:      $(\boldsymbol{y}^{(m)}, w^{(m)}, \alpha^{(m)}) \leftarrow (\varepsilon, 1, \texttt{true})$
4:    while $\exists m \in 1 \dots M : \alpha^{(m)}$ :
5:      for $m = 1 \dots M : \neg\alpha^{(m)}$ :          ▷ Incomplete particles
6:        $y' \sim Q(\cdot \mid \boldsymbol{y}^{(m)})$
7:        if $y' = \mathrm{EOS}$ :                         ▷ Complete the particle
8:          $\alpha^{(m)} \leftarrow \texttt{false}$
9:          $w^{(m)} \leftarrow w^{(m)} \cdot \frac{p(y'|\boldsymbol{y}^{(m)})}{Q(y'|\boldsymbol{y}^{(m)})} \varphi(\boldsymbol{y}^{(m)})$
10:       else
11:         $w^{(m)} \leftarrow w^{(m)} \cdot \frac{p(y'|\boldsymbol{y}^{(m)})}{Q(y'|\boldsymbol{y}^{(m)})}$
12:         $\boldsymbol{y}^{(m)} \leftarrow \boldsymbol{y}^{(m)} \cdot y'$
13:   $\widehat{Z} \leftarrow \frac{1}{M} \sum_{m=1}^{M} w^{(m)}$
14:   $\widehat{P}(\boldsymbol{y}) \leftarrow \widetilde{W}^{-1} \sum_{m=1}^{M} w^{(m)} \mathbb{1}\{\boldsymbol{y} = \boldsymbol{y}^{(m)}\}$
15:   return $(\widehat{Z}, \widehat{P})$
```

poorly because it may sample many particles that show early signs of being poor samples. Thus, rather than sampling complete strings from the proposal $Q$, we seek to *incrementally* evaluate and prioritize samples as they evolve.[10] **Sequential Monte Carlo** (**SMC**; Doucet et al., 2001) augments importance sampling with two key ingredients: *shaping* and *resampling*.

**Shaping.** The idea behind shaping is that while we are generating a string, we should assess the quality of the partially completed particle with respect to the target posterior distribution using a shaping function. A **shaping function**[11] $\psi \colon \Sigma^* \to \mathbb{R}_{\geq 0}$ approximates the expected future potential $\psi^*(\boldsymbol{y})$ of the partially completed string $\boldsymbol{y}$:

$$\psi(\boldsymbol{y}) \approx \psi^*(\boldsymbol{y}) \overset{\text{def}}{=} \mathop{\mathbb{E}}_{\boldsymbol{Y} \sim p} [\varphi(\boldsymbol{Y}) \mid \boldsymbol{Y} \succeq \boldsymbol{y}] \quad (5)$$

where $\boldsymbol{Y} \succeq \boldsymbol{y}$ denotes the event that the string-valued random variable $\boldsymbol{Y}$, distributed according to to the language model $p$, has $\boldsymbol{y}$ as a prefix. Note that the *exact* conditional probability of $y'$, a token or EOS, given the prefix $\boldsymbol{y}$ under the posterior distribution is in fact

$$\psi^*(y' \mid \boldsymbol{y}) = \frac{\psi^*(\boldsymbol{y} \cdot y')}{\psi^*(\boldsymbol{y})}, \quad (6)$$

and $\psi^*(\varepsilon) = Z$. The optimal proposal distribution $Q^*$ is $Q^*(y \mid \boldsymbol{y}) = \psi^*(y' \mid \boldsymbol{y})$, as it draws exact samples according to auto-regressive factorization of the posterior distribution, i.e., $P(y' \mid \boldsymbol{y})$.[12]

Unfortunately, computing $\psi^*$ is, in general, intractable; however, SMC methods allow for approximating $\psi^*$ with a shaping function $\psi$, provided they are **admissible** with respect to the target $\varphi$, i.e., $\forall \boldsymbol{y}, \boldsymbol{y}' \in \Sigma^* \colon p(\boldsymbol{y})\, \psi(\boldsymbol{y}) = 0 \implies \varphi(\boldsymbol{y} \cdot \boldsymbol{y}') = 0$. This condition ensures the particles cannot be incorrectly killed off (i.e., assigned weight zero) by the shaping function. We discuss our proposed approximation in §4.

**Algorithm.** SMC reallocates computational resources to the more promising particles based on their shaped weights such that many of the nice statistical properties of the importance sampling method are preserved. Below, we provide a brief conceptual overview of the SMC algorithm, but refer to the pseudocode in Alg. 2 for a complete technical description. The algorithm begins by initializing $M$ identical particles. Each $1 \leq m \leq M$ particle is represented by a string $\boldsymbol{y}^{(m)}$, a weight $w^{(m)}$, and a boolean $\alpha^{(m)}$ that indicates if it has been completed (i.e., reached EOS). The algorithm runs for $N$ steps, where $N$ is the maximum number of tokens we are willing to consider under the posterior distribution.[13] For each particle $m$, while $\alpha^{(m)}$ is true, its weight $w^{(m)}$ is a shaped estimate; however, once the particle is complete $\alpha^{(m)}$ is false, the weight will equal that of importance sampling (i.e., $\varphi(\boldsymbol{y}^{(m)})$), as the product of the shaped estimates from the first step to this final step cancel each other out.[14]

At each step, each particle $\boldsymbol{y}^{(m)}$ is extended by an additional token by sampling from $y' \sim Q(\cdot \mid \boldsymbol{y}^{(m)})$. If $y'$ is EOS, it is treated specially by updating its $\alpha^{(m)}$ and finalizing its weight $w^{(m)}$; otherwise, we update $\boldsymbol{y}^{(m)}$ by appending the new token $y'$ to it and we update its weight $w^{(m)}$ by multiply the shaping ratio, i.e., $\frac{\psi(\boldsymbol{y}^{(m)} \cdot y')}{\psi(\boldsymbol{y}^{(m)})}$.

Once all particles have advanced to the next step, we may *resample* (bootstrap) the particles, i.e., sample $M$ samples with replacement proportionally to their weights ($w^{(m)}$). The resampled set of $M$ particles replaces the existing particles. We set the weights of the resampled particles equal to the average weight, as this choice preserves unbiased-

[10]Therefore, we assume that the proposal $Q$ factors the same way that the prior $p$ does Eq. (1), i.e., it provides efficient methods to evaluate conditional probabilities $Q(y' \mid \boldsymbol{y})$.

[11]Note that Zhao et al. (2024) call them *twisting* functions.

[12]Note that the shaping function is akin to a *heuristic* function in search (Pearl, 1984); however, the specifics of what makes a heuristic function admissible (i.e., correct to use) differ from those that make a shaping function admissible. We also note that the exact shaping function $\psi^*$ is closely related to the backward probabilities of hidden Markov models.

[13]In our setting, we know the specific number of words that the string must have, so we use that value for $N$.

[14]Consider $\boldsymbol{y}^{(m)} = y_1 y_2 \cdots y_{N-1} y_N$, its final weight is $\psi(\varepsilon) \frac{\psi(y_1)}{\psi(\varepsilon)} \frac{\psi(y_1 y_2)}{\psi(y_1)} \cdots \frac{\psi(y_1 \cdots y_N)}{\psi(y_1 \cdots y_{N-1})} \frac{\varphi(y_1 \cdots y_N)}{\psi(y_1 \cdots y_N)} = \varphi(\boldsymbol{y})$.

**Algorithm 2** Sequential Monte Carlo

```
1:  procedure SMC($p, Q, \varphi, \psi, M, \tau$)
2:    for $m = 1 \ldots M$ :                                   ▷ M number of samples
3:      $(\boldsymbol{y}^{(m)}, w^{(m)}, \alpha^{(m)}) \leftarrow (\varepsilon, \psi(\varepsilon), \texttt{true})$
4:    while $\exists m \in 1 \ldots M : \alpha^{(m)}$ :
5:      for $m = 1 \ldots M : \neg\alpha^{(m)}$ :               ▷ Incomplete particles
6:        $y' \sim Q(\cdot \mid \boldsymbol{y}^{(m)})$
7:        if $y' = \text{EOS}$ :                                ▷ Complete the particle
8:          $\alpha^{(m)} \leftarrow \texttt{false}$
9:          $w^{(m)} \leftarrow w^{(m)} \frac{p(y' \mid \boldsymbol{y}^{(m)})\varphi(\boldsymbol{y}^{(m)})}{Q(y' \mid \boldsymbol{y}^{(m)})\psi(\boldsymbol{y}^{(m)})}$     ▷ Final
10:       else
11:         $w^{(m)} \leftarrow w^{(m)} \frac{p(y' \mid \boldsymbol{y}^{(m)})\psi(\boldsymbol{y}^{(m)} \cdot y')}{Q(y' \mid \boldsymbol{y}^{(m)})\psi(\boldsymbol{y}^{(m)})}$     ▷ Shaped
12:         $\boldsymbol{y}^{(m)} \leftarrow \boldsymbol{y}^{(m)} \cdot y'$
13:     $(\boldsymbol{y}^{(\cdot)}, w^{(\cdot)}, \alpha^{(\cdot)}) \leftarrow$ RESAMPLE($\boldsymbol{y}^{(\cdot)}, w^{(\cdot)}, \alpha^{(\cdot)}, \tau$)
14:   $\widehat{Z} \leftarrow \widetilde{W}/M$
15:   $\widehat{P}(\boldsymbol{y}) \leftarrow \widetilde{W}^{-1} \sum_{m=1}^{M} w^{(m)} \mathbb{1}\{\boldsymbol{y} = \boldsymbol{y}^{(m)}\}$
16:   return $(\widehat{Z}, \widehat{P})$
17: procedure RESAMPLE($\boldsymbol{y}^{(\cdot)}, w^{(\cdot)}, \alpha^{(\cdot)}, \tau$)
18:   $\widetilde{W} \leftarrow \sum_{m=1}^{M} w^{(m)}$
19:   $\widehat{M} \leftarrow \widetilde{W}^2 / \left(\sum_{m=1}^{M} \left(w^{(m)}\right)^2\right)$
20:   if $\widehat{M} < \tau \cdot M$ :                                ▷ Resample
21:     $\overline{\boldsymbol{y}}^{(\cdot)} \leftarrow \boldsymbol{y}^{(\cdot)}$; $\overline{w}^{(\cdot)} \leftarrow w^{(\cdot)}$          ▷ Temporary copy
22:     for $m = 1 \ldots M$ :
23:       $R \sim \text{Categorical}(\widetilde{W}^{-1} \langle \overline{w}^{(1)}, \ldots, \overline{w}^{(M)} \rangle)$
24:       $(\boldsymbol{y}^{(m)}, w^{(m)}, \alpha^{(m)}) \leftarrow (\overline{\boldsymbol{y}}^{(R)}, \widetilde{W}/M, \alpha^{(R)})$
25:   return $(\boldsymbol{y}^{(\cdot)}, w^{(\cdot)}, \alpha^{(\cdot)})$
```

ness of $\widehat{Z}$.[15] Most importantly, the lower-weight particles are less likely to be selected relative to the higher-weight particles. This means the more promising particles, i.e., those with higher weights, are more likely to be replicated, and the later steps will extend their replicas.

Notice, however, that, unlike importance sampling, the samples produced by this method are no longer independent due to their shared history. This has the downside of producing a low-diversity sample, but it has the upside that the samples produced tend to be more representative of the posterior distribution. To mitigate the issue of overly dependent samples, the resampling step is typically not performed at every step, but only when necessary. A common criterion for deciding when to resample is the **effective sample size** $\widehat{M}$ (defined on line 19)[16] Resampling is triggered only when $\widehat{M}$ falls below a predefined threshold $\tau M$.

[15]This resampling strategy is known as *multinomial resampling* in the SMC literature (Doucet et al., 2001).

[16]$\widehat{M}$ translates a set of weighted samples into an equivalent number of unweighted samples in relation to the variance reduction. For ordinary, unweighted Monte Carlo, $M$ samples reduce the variance of the estimator by roughly a factor of $1/M$. In contrast, for importance sampling, the variance reduction is roughly reduced by $1/\widehat{M}$ (Martino et al., 2017).

**Guarantees.** SMC has the same guarantees as importance sampling. We also note that shaping is *ignored* if resampling is disabled by setting $\tau = 0$, making the SMC algorithm's samples equivalent to importance sampling, and the procedure equivalent to sequential importance sampling.

## 3 Tetratagger Likelihood

In this section, we describe the syntactic likelihood model $\varphi(\mathbf{t} \mid \boldsymbol{y})$, based on tetratagger. A tetratagger is a neural constituency parser that works by assigning two tags to each word in the sentence, except for the last word, which is assigned one tag. The tags assigned to a word represent how this word and its parent (which is an intermediate node) are situated in the constituency tree. We denote the set of tags by $\mathcal{T}$, which consists of *four* tag types,[17] hence the name *tetra*tagger. Given a string of $L$ words $\boldsymbol{y} = x_1 x_2 \cdots x_L$,[18] its corresponding binarized constituency contains $2L-1$ nodes.[19] Tetratagger encodes the syntactic structure by assigning two tags to each word in the sequence. Therefore, $\varphi$ models two conditional probability distributions over tags $\mathcal{T}$ per word, except the last word to which we only assign one tag. For notational convenience, we assign an extra dummy tag to the last word with probability 1, such that now each word is assigned exactly two tags. Given $\boldsymbol{y}$, the probability of a tag sequence $\mathbf{t}$ under $\varphi$ is

$$\varphi(\mathbf{t} \mid \boldsymbol{y}) \overset{\text{def}}{=} \prod_{\ell=1}^{L} \varphi(t_{2\ell-1} \mid \boldsymbol{y}) \varphi(t_{2\ell} \mid \boldsymbol{y}) \tag{7}$$

**Modeling.** The Tetratagger's conditional distributions $\varphi(t_{2\ell-1} \mid \boldsymbol{y})$ and $\varphi(t_{2\ell} \mid \boldsymbol{y})$ are defined by a pair of linear transformations on top of the existing transformer-based LM. More specifically, we first pass $\boldsymbol{y}$ to a language model and extract the $d$-dimensional latent representation $\mathbf{h}(x_\ell \mid \boldsymbol{y}) \in \mathbb{R}^d$ of $x_\ell$ from the last layer of the language model. Next, we apply each transformation $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{\mathcal{T} \times d}$ to this representation. Finally, we apply softmax to

[17]Two of the tag types encode information about the leaf nodes in the constituency tree and the other two encode information about the internal nodes. The four tags can be enhanced with labels for labeled trees; See (§3.5; Kitaev and Klein, 2020).

[18]Here, we slightly abuse the notation $\boldsymbol{y}$ to represent both a sequence of tokens and a sequence of words, allowing flexibility in how we segment the sequence.

[19]Binarization transforms a constituency tree to a binary tree, where each node has at most two children. We refer the reader to (§3.5; Kitaev and Klein, 2020) for an explanation of how Tetratagger handles non-binary trees.

predict the distribution over the tags:

$$\varphi(t_{2\ell-1} \mid \boldsymbol{y}) \overset{\text{def}}{=} \text{softmax}\big(\mathbf{U}\,\mathbf{h}(x_\ell \mid \boldsymbol{y})\big)_{t_{2\ell-1}} \quad (8a)$$

$$\varphi(t_{2\ell} \mid \boldsymbol{y}) \overset{\text{def}}{=} \text{softmax}\big(\mathbf{V}\,\mathbf{h}(x_\ell \mid \boldsymbol{y})\big)_{t_{2\ell}} \quad (8b)$$

Given a dataset of strings and their corresponding ground-truth tetratags, i.e., a dataset of $(\boldsymbol{y}, \mathbf{t})$ pairs, we learn $\mathbf{U}$ and $\mathbf{V}$ by maximizing the conditional log-likelihood of the dataset.

**Tokens vs. words.** While the language model in Eq. (8a) is assumed to represent *words* in a string, language models operate over *tokens* and not words. Therefore, following Kitaev and Klein (2020), we set $\mathbf{h}(x_\ell \mid \boldsymbol{y})$ to be the representation of the *last token* of $x_\ell$ extracted from the last layer of the language model.

## 4 Autoregressive Tetratagger Shaping

We now introduce an *autoregressive* Tetratagger, which we use as a shaping function ($\psi$). We first transform the desired constituency tree to a sequence of tags $\mathbf{t} = t_1 t_2 \ldots, t_{2L-1}$, where $L$ is the number of words in the tree's yield $\boldsymbol{y}$. For simplicity, we will assume for now that each word consists of a single token (i.e., $L = N$); We will explain at the end of this section how to support multi-token words. We assume that the Tetratagger is using an autoregressive LM backbone by factorizing Eq. (7). Thus, we define the autoregressive tetratagger probability as

$$\psi(\mathbf{t} \mid \boldsymbol{y}) \overset{\text{def}}{=} \prod_{n=1}^{N} \psi(t_{2n-1} \,|\, \boldsymbol{y}_{\leq n})\psi(t_{2n} \,|\, \boldsymbol{y}_{\leq n}), \quad (9)$$

The crucial difference from the likelihood $\varphi$ (Eq. (7)) is that when we predict the two tags at position $n$, we only have access to the string up to that point ($\boldsymbol{y}_{\leq n}$), rather than the complete string ($\boldsymbol{y}$). This makes $\psi$ an effective shaping function as it can be efficiently evaluated as the words of the sentence are being generated left to right.[20] While one might argue that bi-directional information may be critical for accurately predicting syntactic structure, we found that our autoregressive tetratagger is a decent parser (see §5.2). Note that at the final step where we generate the EOS token (Line 9), we calculate the weights by using the likelihood $\varphi(\mathbf{t} \mid \boldsymbol{y})$, as it is now a complete sentence.[21] Lastly, we mention that $\psi$ is parameterized in the same way as $\varphi$ (see Eq. (8a)); thus, it can be trained in precisely the same way. We provide additional training details in §5.2.

[20]The ratio $\frac{\psi(\boldsymbol{y}^{(m)} \cdot y')}{\psi(\boldsymbol{y}^{(m)})}$ on line 11 at the $n$-th step then simplifies to $\psi(t_{2n-1} \,|\, \boldsymbol{y}^{(m)} \cdot y')\psi(t_{2n} \,|\, \boldsymbol{y}^{(m)} \cdot y')$.

| Metrics | 0-shot | 5-shot | 1-shot (gold) |
|---|---|---|---|
| correct length | 42.05 | 43.37 | 65.89 |
| exact match | 17.55 | 19.54 | 31.78 |
| structure match | 20.53 | 25.16 | 44.37 |
| F1 | 47.23 | 53.27 | 70.17 |
| log $\varphi$ | $-\infty$ | $-\infty$ | $-17.41$ |

Table 1: Results of GPT4 with 0-shots, 5-shots and 1-shot of gold example on our evaluation dataset. We observe GPT4 does fails to generate sentences with the correct syntactic structure in most cases, and achieves F1 score of only 53.27 with 5 examples. Even when the model has access to the ground-truth sentence (1-shot (gold)) it only achieves F1 score of 70.17 on the dataset.

**Multi-token words.** We again highlight the fact that Tetratagger assigns tag probabilities to the last token of each word. Therefore, in cases where the sampled token from the proposal LM is not the last token of the word, we keep sampling from the LM until we hit the last token. To detect word boundaries, we use a heuristic algorithm, where we decode the next token and observe whether the next token starts a new word.

## 5 Experiments

**Dataset.** We compile a dataset of constituency trees from a subset of human-generated sentences originally produced by Chen et al. (2019). The dataset is built using paraphrase pairs from the ParaNMT dataset (Wieting and Gimpel, 2018). Each data point consists of a semantic input, a syntactic input, and a reference. The reference shares the same semantic content as the semantic input and mirrors the syntactic structure of the syntactic input. For our experiments, we use the references, which are human-generated sentences that exhibit syntactic variation. We then use the Berkeley Neural Parser (Kitaev et al., 2019; Kitaev and Klein, 2018) to parse these sentences to constituency trees. The leaf nodes of the trees that correspond to the words of the initial sentences are replaced with question marks ("?"), as depicted in Fig. 1. Our

[21]Note that if we sample a different token at the $(N+1)^{\text{th}}$ step, we instead force the EOS token to be generated since our generated sentence has a length-constraint.

dataset consists of a diverse set of trees as shown in Tab. 2.

**Prompt formulation.** For instruction-tuned models, we first experimented with various prompt formulations and ultimately selected the best-performing one shown in App. A.4. To include the constituency trees in this prompt for few-shot settings, we need to linearize the tree. This is done using the bracketing representation of the trees and by replacing the leaf nodes with question marks as mentioned above. Note that for non-instruction models, the prompt is empty.

**Evaluation.** For evaluation of the generated sentences, we parse them to constituency trees and Tetratags. We compute the following metrics:

- **correct length** is the percentage of generated sentences that match the exact word count specified by the constituent tree.
- **exact match** is the percentage of sentences whose generated parse trees match the desired parse trees exactly, both in structure and labels.
- **structure match** is the percentage of sentences whose generated parse trees match the desired parse trees in structure (ignoring labels).
- **F1** is the mean bracketing F1 score comparing the desired constituency trees with the constituency trees of the generated sentences.
- $\log \varphi$ is the median[22] log-likelihood of the original Tetratagger, as defined by Eq. (7) and trained with BERT by Kitaev and Klein (2020) across all generated sentences of dataset.
- $\log p$ is the average log-prior probability of the generated string by the language model used.
- **diversity** (**$n$-gram**) is the average number of distinct $n$-grams in the sentences generated from a given tree, normalized by the total length of all sentences.

### 5.1 `GPT4` Performance

We first examine how challenging it is for state-of-the-art instruction-tuned language models to follow specific syntactic structures. Specifically, we assess how effectively `GPT4` can generate sentences that match the target syntactic structure, both in zero-shot and few-shot settings. The results are reported in Tab. 1. First, we observe that only 42.05% and 43.37% of `GPT4` generations have the desired length using 0 and 5 shots, respectively. By manually inspecting the generations, we noticed that `GPT4` tends to add adjectives, adverbs, or commas to sentences even though the corresponding syntax does not include one. For example, for the syntax `(S (NP (EX ?)) (VP (VBZ ?) (ADVP (RB ?)) (NP (DT ?) (NN ?))))`, `GPT4` produces the sentence "`There is clearly a big problem`". This leads to sentences with incorrect lengths.

In all experimental setups, as reflected in Tab. 1, `GPT4` struggles to generate sentences with the desired structure. `GPT4` achieves F1 score of 47.23 and 53.27 with 0-shot and 5-shot respectively. To explore how far we can improve upon 0-shot and 5-shot performance, we experiment with including an example of a sentence with the exact desired constituency tree in the prompt and evaluate the model's performance in a 1-shot (gold) setting. Surprisingly, even with the gold sentences included in the prompt, the model fails to consistently copy the sentence from the prompt into the output, achieving only an F1 score of 70.17. Overall, the results indicate that generating sentences with a specific syntactic structure remains a challenging task, even for state-of-the-art language models.

### 5.2 Training Tetratagger Shaping Functions

We train Tetratagger using the `GPT2` and `Llama3-8B` models as the language model backbone. We later use these taggers to control the decoding process. Importantly, we match the LM backbone of the tagger with the language model used for generating text. This is essential to align the tokenization scheme of Tetratagger with the language model we use to generate text. Both Tetratagger models were trained for two epochs on the Penn Treebank dataset (Marcus et al., 1993), using the cross-entropy loss.

We report the accuracy of our autoregressive Tetratagger in predicting correct tags for leaf nodes (Leaf Acc.) and internal nodes (Internal Acc.). Furthermore, we convert the sequence of predicted tags to constituency trees and compute the bracketing F1 score. Our autoregressive Tetrataggers with `GPT2-large` and `Llama3-8B` achieve an F1 score of 68.75 and 71.79, Leaf Acc. of 93.52 and 94.45, and Internal Acc. of 81.53 and 82.96 respectively, showing that we can achieve a reasonable performance even though we do not include any bidirectional information and thus our tagger does not have access to the whole sequence when predicting each tag.

[22]We compute the log of the median of the exponentiated $\log \varphi$ values, rather than using the mean, to prevent $-\infty$ values from skewing the results.

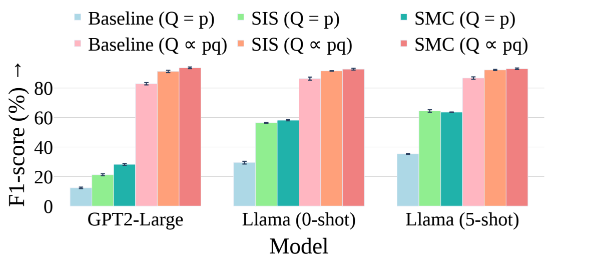
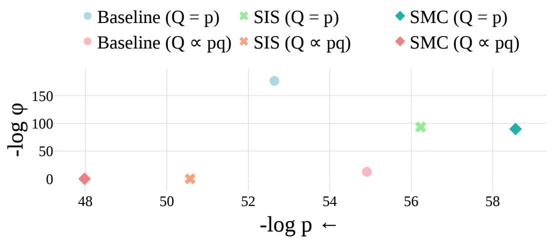
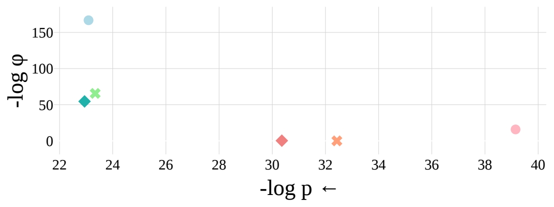


Figure 2: F1 score across all models and methods using the two proposal distributions, $Q = p$ and $Q \propto p\,q$. Note that our baseline is sampling one sentence with $N$ words directly from $p$. We observe that SMC achieves the largest F1-score in most cases, while the choice of proposal distribution plays a crucial role in further boosting the syntactic score.



(a) GPT2-large



(b) Llama3-Instruct (0-shot)

Figure 3: $-\log\varphi$ and $-\log p$ for different methods, proposal distributions $Q = p$ and $Q = pq$ and GPT2-large and Llama3-Instruct (0 shots) models. Our approach does not compromise fluency for syntactic consistency.

## 5.3 Choice of Proposal Distribution

In the experiments, we explore two different options for the proposal distribution. The first is the prior distribution itself:

$$Q(y' \mid \boldsymbol{y}_{<n}) = p(y' \mid \boldsymbol{y}_{<n}) \tag{10}$$

While simple, the prior may not effectively approximate the posterior $P$, especially when it does not contain much information about the control. Therefore, we also consider an alternative proposal that incorporates syntactic information via a bigram model conditioned on part-of-speech tags:

$$Q(\boldsymbol{y}' \mid \boldsymbol{y}_{<n}) \propto p(y'_1 \mid \boldsymbol{y}_{<n})q(\boldsymbol{y}' \mid z_n z_{n+1}), \tag{11}$$

where $y'_1$ is the first token of the word $\boldsymbol{y}'$,[23] $z_n$ is the part-of-speech tag at position $n$ in the syntax tree, and $q$ denotes the probability distribution derived from the bigram model trained on the Penn Treebank dataset.[24] We chose a bigram model over part-of-speech tags rather than a more sophisticated model, as we found that it adequately captured syntax information without overfitting to its training corpus. This approach aims to provide a more informed proposal by better approximating the expected future potential in conjunction with the shaping function, which already accounts for the tokens generated so far.

[23]This choice was made for efficiency, as evaluating the probability of all possible words is prohibitive.

[24]More specifically, the bigram model was trained to predict the $n^{\text{the}}$ word of a sentence $\boldsymbol{y}_n$ given the part-of-speech tags $z_n$ and $z_{n+1}$. Note that words may be multiple tokens long.

## 5.4 Controlled Generation Results[25]

We repeat each experiment five times for GPT2-large and two times for Llama3-8B models. The experiments were conducted using $\tau = 0.25$ (for SMC), $M = 20$ when $Q = p$ and $M = 6$ when $Q \propto p\,q$, as the latter provides a more informed proposal distribution, allowing us to achieve good performance with fewer particles (See §5.5). Note that as $\varphi$ we use the Tetratagger trained by Kitaev and Klein (2020) using the BERT model (Devlin et al., 2019), which achieves 95.4 F1 score. We evaluated all sampling algorithms using language models of varying sizes.

The plot in Fig. 2 shows that using Eq. (10) as the proposal distribution improves F1 scores considerably with both SIS and SMC across all models, more than doubling performance with SMC (from 12.31 to 28.26 with GPT2-large and from 29.41 to 58.18 with Llama3-Instruct (0-shot). While SMC performs slightly worse than SIS in 5-shot settings, it generally yields higher gains. These findings also highlight the critical role of proposal distribution selection, especially for non-instruction-tuned models. Sampling directly from the prior fails to produce high-likelihood samples, as obtaining a high-quality sample requires $M \propto 1/Z$, which is impractical when $Z$ is small. This is further supported by results using $Q \propto p\,q$, reaching

[25]We refer the reader to App. B for additional experimental results and supplementary tables.

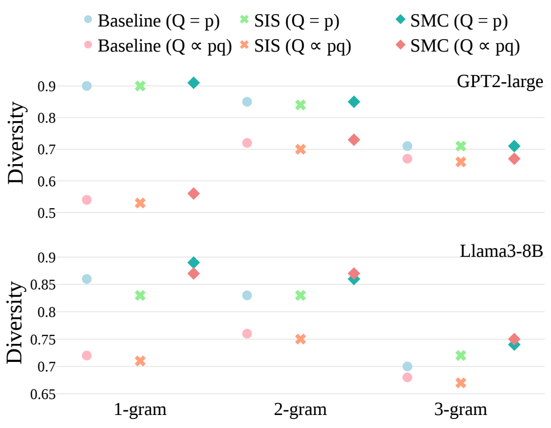


Figure 4: Diversity metric across methods and `GPT2-large` and `Llama3-Instruct` (0 shots) models. $Q \propto p\,q$ proposal decreases diversity of generated sentences, especially in the case of `GPT2-large` model.

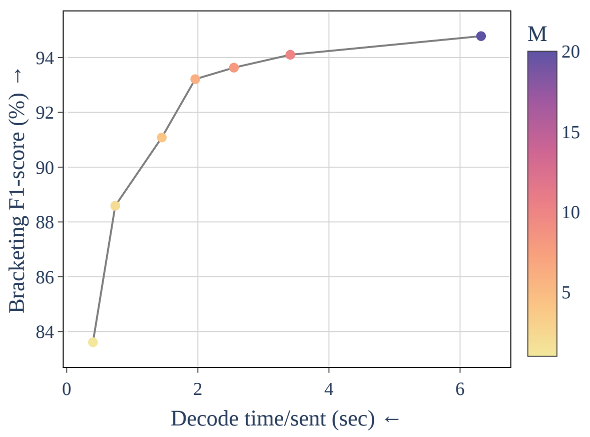


Figure 5: F1-score vs. SMC time/sentence for varying numbers of particles $M$ with $Q \propto p\,q$ for `GPT2-large`.

up to 93% F1-score, making smaller models competitive to larger ones. However, this comes at the cost of diversity, particularly with `GPT2-large`, where vocabulary restrictions imposed by the bigram model lead to repetitive outputs (See Fig. 4).

We also evaluate syntactic quality and text fluency using $-\log \varphi$ and $-\log p$, as shown in Fig. 3. These metrics capture different aspects of structural fidelity than F1 alone. When using the language model as the proposal distribution, SIS and SMC improve $\log \varphi$, achieving a $1/3$ reduction in both `GPT2-large` and `Llama3-Instruct`, but slightly worsen $\log p$, indicating a trade-off. The degradation in $\log p$ is more pronounced with `GPT2-large` where $-\log p$ increases by 5 points, while in `Llama3-Instruct` it is marginal. This occurs because our method places greater emphasis on syntactic accuracy; however, the slight degradation shows that it does not compromise the language model's fluency. Notably, when using $Q \propto p\,q$, $-\log \varphi$ reaches values near zero (i.e., the syntax is nearly a perfect match) with both SMC and SIS, while $-\log p$ simultaneously improves. We attribute this to the vocabulary restriction, which may introduce noise when the generation is left uncontrolled, but is effectively corrected when our method is applied.

### 5.5 Decoding Time

Fig. 5 illustrates the relationship between decoding time per sentence and the corresponding bracketing F1 score, for varying values of $M$ using SMC with $Q \propto p\,q$ on the `GPT2-large` model.[26] As $M$ increases, there is a clear improvement in the syntactic quality of the generated sentences, with the F1 score rising from 83% to nearly 95%, supporting our hypothesis. However, beyond 6 particles, the improvements become marginal.

## 6 Conclusion

In this paper, we introduced a sampling algorithm designed to control the syntactic structure of text generated by language models. Our method, based on sequential Monte Carlo, leverages a parsing-as-tagging framework, guiding the generation process by incorporating syntactic taggers. At each step of text generation, the algorithm samples $M$ particles from a proposal distribution and assigns weights to them using a shaping function that assesses the syntactic structure of the partial sequences generated. Our sampling algorithm ensures that both linguistic content and syntactic correctness are taken into account. Our experiments with `GPT2` and `Llama3-8B` models demonstrate that our approach achieves large improvements in generating text that aligns with target syntactic structures. These results highlight the potential of combining sampling algorithms with tagging frameworks to enhance the syntactic controllability of language models.

## Acknowledgments

The authors would like to thank Ben LeBrun, Manuel de Prada Corral, and Robin Shing Moon Chan for valuable feedback and discussions. Afra

[26]Note that in addition to the time required for sampling from the proposal distribution $Q$, all candidate strings must also be evaluated by the tetrataggers $\psi$ and $\varphi$.

Amini is supported by the ETH AI Center doctoral fellowship.

## Limitations

**Pruned-height trees.** Our method cannot be applied to constituent trees with a pruned height. Since the algorithm relies on full tree structures to assign each token to its corresponding tags, pruning the tree height disrupts this mapping, making the approach unsuitable for such cases.

**No fault tolerance.** Our algorithm restricts the length of the generated sentence to the desired number of words defined by the constituent tree. While this ensures that the generated output conforms to specific length constraints, there are instances where a fully coherent sentence is not formed by the time the algorithm finishes. This issue can arise when, in a generation step, the appropriate token does not appear within the $M$ tokens sampled or when the Tetratagger model assigns incorrect tags to words. In such scenarios, if the algorithm could complete the full sentence, even with some syntactic errors, we might see the results would better align with the desired syntax, leading to improved outcomes.

**Language.** In this paper, we focus on English, mainly due to better data availability and improved model performance. However, exploring how this approach generalizes to other languages with diverse syntactic structures is an interesting direction for future work.

## Ethical considerations

We acknowledge that our approach may introduce unexpected biases or inaccuracies, which could result in outputs that are factually incorrect or contextually inappropriate. This risk arises because our method modifies the probability distribution of a language model to align the generated sentences with specific syntactic structures, which could unintentionally amplify the risk of producing harmful outputs or misinformation. Consequently, users must exercise caution when applying our method, particularly in sensitive or public-facing contexts, to mitigate the potential for unintended negative consequences.

## References

Afra Amini and Ryan Cotterell. 2022. On parsing as tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Afra Amini, Li Du, and Ryan Cotterell. 2023. Structured Voronoi sampling. In *Proceedings of the Conference on Neural Information Processing Systems*.

Dhananjay Ashok and Barnabas Poczos. 2024. Controllable text generation in the instruction-tuning era. *Computing Research Repository*, arXiv:2405.01490.

Bruce K. Britton, Shawn M. Glynn, Bonnie J Meyer, and M. J. Penland. 1982. Effects of text structure on use of cognitive capacity during reading. *Journal of Educational Psychology*, 74(1).

Sourav Chatterjee and Persi Diaconis. 2017. The sample size required in importance sampling. *Preprint*, arXiv:1511.01437.

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019. Controllable paraphrase generation with a syntactic exemplar. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Arnaud Doucet, Nando De Freitas, and Neil James Gordon. 2001. *Sequential Monte Carlo Methods in Practice*. Springer.

Richard Futrell, Ethan Wilcox, Takashi Morita, Peng Qian, Miguel Ballesteros, and Roger Levy. 2019. Neural language models as psycholinguistic subjects: Representations of syntactic state. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

David Kauchak. 2013. Improving text simplification language modeling using unsimplified text data. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1537–1546, Sofia, Bulgaria. Association for Computational Linguistics.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Nikita Kitaev and Dan Klein. 2020. Tetra-tagging: Word-synchronous parsing with linear-time inference. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Ben Krause, Akhilesh Deepak Gotmare, Bryan McCann, Nitish Shirish Keskar, Shafiq Joty, Richard Socher, and Nazneen Fatema Rajani. 2021. GeDi: Generative discriminator guided sequence generation. In *Findings of the Association for Computational Linguistics*.

Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka. 2023. Sequential Monte Carlo steering of large language models using probabilistic programs. *Preprint*, arXiv:2306.03081.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4.

Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. 2021. DExperts: Decoding-time controlled text generation with experts and anti-experts. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing*.

Llama Team. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

João Loula, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell. 2025. Syntactic and semantic control of large language models via sequential Monte Carlo. In *Proceedings of The International Conference on Learning Representations*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2).

Luca Martino, Víctor Elvira, and Francisco Louzada. 2017. Effective sample size for importance sampling based on discrepancy measures. *Signal Processing*, 131.

OpenAI. 2024. GPT-4 technical report. *Preprint*, arXiv:2303.08774.

Judea Pearl. 1984. *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley series in artificial intelligence. Addison-Wesley.

Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable code generation from pre-trained language models. In *International Conference on Learning Representations*.

Alec Radford, Jeff Wu, Rewon Child, D. Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Adithya Renduchintala, Rebecca Knowles, Philipp Koehn, and Jason Eisner. 2016. Creating interactive macaronic interfaces for language learning. In *Proceedings of Association for Computational Linguistics System Demonstrations*.

Timo Schick, Sahana Udupa, and Hinrich Schütze. 2021. Self-diagnosis and self-debiasing: A proposal for reducing corpus-based bias in NLP. *Transactions of the Association for Computational Linguistics*, 9.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen Jr, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Jiao Sun, Yufei Tian, Wangchunshu Zhou, Nan Xu, Qian Hu, Rahul Gupta, John Wieting, Nanyun Peng, and Xuezhe Ma. 2023. Evaluating large language models on controlled generation tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. 2024. SynCode: LLM generation with grammar augmentation. *Preprint*, arXiv:2403.01632.

Robert Vacareanu, George Caique Gouveia Barbosa, Marco A. Valenzuela-Escárcega, and Mihai Surdeanu. 2020. Parsing as tagging. In *Proceedings of the Language Resources and Evaluation Conference*.

Sandra H. Vos, Thomas C. Gunter, Herbert Schriefers, and Angela D. Friederici. 2001. Syntactic parsing and working memory: The effects of syntactic complexity, reading span, and concurrent load. *Language and Cognitive Processes*, 16(1).

John Wieting and Kevin Gimpel. 2018. ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Wei Xu, Chris Callison-Burch, and Courtney Napoles. 2015. Problems in current text simplification research: New data can help. *Transactions of the Association for Computational Linguistics*, 3:283–297.

Kevin Yang and Dan Klein. 2021. FUDGE: Controlled text generation with future discriminators. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Baker Grosse. 2024. Probabilistic inference in language models via twisted sequential Monte Carlo. In *Proceedings of the International Conference on Machine Learning*.

## A Experimental Details

### A.1 Models

We experimented with different models of different sizes: `GPT2-large` (774M parameters), `Llama3-8B` (8B parameters) for token generation and as the LM backbone of Tetratagger. For `GPT4`, we used the OpenAI API[27] to generate sentences. All other language models utilized in this research are sourced from HuggingFace (Wolf et al., 2020) and are subject to specific licences that govern their use and distribution in the research domain. Specifically, we used `GPT2-large`[28] under MIT License and `Llama3-8B`[29] and `Llama3-Instruct`[30] under license META LLAMA 3 COMMUNITY LICENSE AGREEMENT.

All experiments for `GPT2-large` were conducted on a single `NVIDIA GeForce RTX 4090 GPU` with 24 GiB memory, while for `Llama3-8B` on a single `NVIDIA Tesla V100-SXM2` with 32 GiB memory.

### A.2 Autoregressive Tetratagger

We trained our autoregressive Tetratagger models based on https://github.com/nikitakit/tetra-tagging on the Penn Treebank, as suggested by Kitaev and Klein (2020), a standard benchmark for evaluating syntactic parsing algorithms. Our tag vocabulary consists of 231 labels in total.

For efficiency reasons, to train a Tetratagger with `Llama3-8B` as the LM backbone, we applied four-bit quantization and fine-tuned the model using LoRA (Hu et al., 2022). This allows us to reduce the number of trainable parameters to $1.06\%$ of the total model parameters.

### A.3 Evaluation Dataset

Our evaluation dataset consists of 301 syntax trees varying on number of nodes, height, and labels. Dataset statistics are provided in Tab. 2. All constituent trees utilized in our experiments were represented in the bracketing representation, which aligns with standard practices in syntactic parsing. For generating the tag

| | |
|---|---|
| Number of trees | 301 |
| Mean height | 7.11 |
| Max height | 18.00 |
| Mean leaf nodes | 8.51 |
| Max leaf nodes | 20.00 |
| Mean tree size | 24.42 |

Table 2: Statistics for the dataset used for evaluation of our method.

sequences of our syntax trees, we utilized the deterministic algorithm of the pre-trained Tetratagger (§3.1; Kitaev and Klein, 2020). In Fig. 6 we give an example of a tag sequence for a specific tree. For parsing the generated sentences into constituent trees, we used the Berkeley Neural Parser[31] through spaCy.[32] The predicted syntactic trees were then compared to the ground truth syntax trees in our dataset.

**Tree**: `(S (NP (EX There)) (VP (VBZ is) (ADVP (RB always)) (NP (DT a) (NN chance))))`
**Tag Sequence**: `['l/NP', 'L/S', 'l', 'R/VP', 'l/ADVP', 'R', 'l', 'R/NP', 'r']`

Figure 6: Example of a tag sequence given a tree generated by the deterministic algorithm of Kitaev and Klein (2020)

[27] https://openai.com/policies/terms-of-use/
[28] https://huggingface.co/openai-community/gpt2-large
[29] https://huggingface.co/meta-llama/Meta-Llama-3-8B
[30] (https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
[31] https://pypi.org/project/benepar/
[32] https://spacy.io/

**Prompt Template**

```
You are a helpful assistant that generates a sentence.
From a given constituency parse tree, output only one
grammatical English sentence that matches the syntactic
structure. Do not include any explanations, preambles,
or quotation marks. Respond with the sentence only.
-------------------------------------------------------
An example is first provided:
Parse tree: {SYNTAX}
Sentence: {SENTENCE}
-------------------------------------------------------
Now generate a sentence for the following tree.
Parse tree: {SYNTAX}
-------------------------------------------------------
Sentence:
```

(a) `Llama3-Instruct`

**Prompt Template**

```
You are a helpful assistant in generating a sentence
from the provided syntactic structure defined by a
constituency parse tree. Please only have the generated
sentence, not its parse, in the response.
-------------------------------------------------------
An example is first provided:
Parse tree: {SYNTAX}
Answer: {SENTENCE}
-------------------------------------------------------
Now generate a sentence for the following parse tree:
Parse tree: {SYNTAX}
-------------------------------------------------------
Sentence:
```

(b) `GPT4`

Figure 7: The prompt used for generating text with `Llama3-Instruct` and `GPT4` models. Note that the exact format of the prompt has been simplified; in particular, special tokens and roles have been omitted to improve readability.

## A.4 Prompt Formulation

After applying different prompt formulations for instruction-tuned models, we used the prompts displayed in Fig. 7 for `Llama3-Instruct` and `GPT4`. For `Llama3-Instruct`, we incorporated the special tokens and roles provided by Llama Team (2024) to achieve the best roles. Moreover, we found the appropriate prompt that would force the language model to generate the output sentence directly in the case of `Llama3-Instruct`. In the case of `GPT4` this was not necessary, since we could post-process the generated sentences to remove any explanation or quotation mark.

## B Full Experimental Results

In Tab. 3 we report our full experimental results (the mean and the standard deviation over the runs). Please refer to §5.4 for more details about the settings of algorithms SIS and SMC. Note that we observed that `Llama3-8B` tends to generate content related to coding or begins its outputs with questions when prompted with the BOS token. This behavior does not help our method to be applied effectively when the prior equals the language model's distribution.

| **Models** | $\log \varphi(\boldsymbol{y}) \uparrow$ | $\log p(\boldsymbol{y}) \uparrow$ | **F1-score** $\uparrow$ | **Diversity** | | |
|---|---|---|---|---|---|---|
| | | | | **1-gram** $\uparrow$ | **2-gram** $\uparrow$ | **3-gram** $\uparrow$ |
| | | | $Q = p$ | | | |
| `GPT2-large` | $-176.80_{\pm 9.17}$ | $-52.64_{\pm 0.33}$ | $12.31_{\pm 0.27}$ | $0.90_{\pm 0.05}$ | $0.85_{\pm 0.06}$ | $0.71_{\pm 0.12}$ |
| `GPT2-large` + SIS | $-93.45_{\pm 4.15}$ | $-56.23_{\pm 0.78}$ | $21.18_{\pm 0.45}$ | $0.90_{\pm 0.06}$ | $0.84_{\pm 0.06}$ | $0.71_{\pm 0.12}$ |
| `GPT2-large` + SMC | $-89.89_{\pm 5.58}$ | $-58.56_{\pm 0.38}$ | $28.26_{\pm 0.39}$ | $0.91_{\pm 0.05}$ | $0.85_{\pm 0.06}$ | $0.71_{\pm 0.12}$ |
| `Llama3-8B` | $-\infty$ | $-62.47_{\pm 1.66}$ | $10.05_{\pm 0.56}$ | $0.77_{\pm 0.23}$ | $0.71_{\pm 0.19}$ | $0.61_{\pm 0.17}$ |
| `Llama3-8B` + SIS | $-53.86_{\pm 1.78}$ | $-79.79_{\pm 1.04}$ | $16.45_{\pm 0.24}$ | $0.92_{\pm 0.09}$ | $0.85_{\pm 0.07}$ | $0.72_{\pm 0.12}$ |
| `Llama3-8B` + SMC | $-64.12_{\pm 1.58}$ | $-117.88_{\pm 0.33}$ | $16.12_{\pm 0.49}$ | $0.91_{\pm 0.09}$ | $0.86_{\pm 0.05}$ | $0.75_{\pm 0.09}$ |
| `Llama3-8B` (0-shot) | $-166.86_{\pm 4.46}$ | $-23.09_{\pm 0.68}$ | $29.41_{\pm 0.88}$ | $0.86_{\pm 0.09}$ | $0.83_{\pm 0.09}$ | $0.70_{\pm 0.12}$ |
| `Llama3-8B` (0-shot) + SIS | $-65.66_{\pm 0.78}$ | $-23.34_{\pm 0.11}$ | $56.42_{\pm 0.11}$ | $0.83_{\pm 0.10}$ | $0.83_{\pm 0.09}$ | $0.72_{\pm 0.12}$ |
| `Llama3-8B` (0-shot) + SMC | $-54.58_{\pm 1.27}$ | $-22.94_{\pm 0.22}$ | $58.18_{\pm 0.18}$ | $0.89_{\pm 0.07}$ | $0.86_{\pm 0.04}$ | $0.74_{\pm 0.06}$ |
| `Llama3-8B` (5-shot) | $-133.95_{\pm 13.20}$ | $-23.12_{\pm 0.17}$ | $35.33_{\pm 0.11}$ | $0.88_{\pm 0.09}$ | $0.84_{\pm 0.08}$ | $0.70_{\pm 0.12}$ |
| `Llama3-8B` (5-shot) + SIS | $-43.94_{\pm 4.62}$ | $-22.71_{\pm 0.36}$ | $64.44_{\pm 0.67}$ | $0.86_{\pm 0.09}$ | $0.84_{\pm 0.09}$ | $0.72_{\pm 0.12}$ |
| `Llama3-8B` (5-shot) + SMC | $-40.89_{\pm 5.51}$ | $-24.02_{\pm 0.09}$ | $63.66_{\pm 0.01}$ | $0.92_{\pm 0.06}$ | $0.87_{\pm 0.04}$ | $0.74_{\pm 0.06}$ |
| | | | $Q \propto p\,q$ | | | |
| `GPT2-large` | $-12.71_{\pm 1.17}$ | $-54.91_{\pm 0.53}$ | $82.89_{\pm 0.61}$ | $0.54_{\pm 0.09}$ | $0.72_{\pm 0.10}$ | $0.67_{\pm 0.12}$ |
| `GPT2-large` + SIS | $-0.0001_{\pm 0.0}$ | $-50.57_{\pm 0.16}$ | $91.22_{\pm 0.66}$ | $0.53_{\pm 0.09}$ | $0.70_{\pm 0.11}$ | $0.66_{\pm 0.13}$ |
| `GPT2-large` + SMC | $-0.0009_{\pm 0.0001}$ | $-47.98_{\pm 0.34}$ | $93.69_{\pm 0.33}$ | $0.56_{\pm 0.09}$ | $0.73_{\pm 0.11}$ | $0.67_{\pm 0.13}$ |
| `Llama3-8B` | $-14.93_{\pm 0.49}$ | $-89.11_{\pm 0.24}$ | $76.49_{\pm 0.88}$ | $0.75_{\pm 0.09}$ | $0.81_{\pm 0.09}$ | $0.71_{\pm 0.12}$ |
| `Llama3-8B` + SIS | $-0.28_{\pm 0.20}$ | $-75.58_{\pm 0.24}$ | $86.50_{\pm 0.11}$ | $0.75_{\pm 0.09}$ | $0.81_{\pm 0.08}$ | $0.71_{\pm 0.12}$ |
| `Llama3-8B` + SMC | $-6.68_{\pm 0.07}$ | $-70.80_{\pm 0.87}$ | $90.18_{\pm 0.18}$ | $0.88_{\pm 0.07}$ | $0.86_{\pm 0.04}$ | $0.73_{\pm 0.08}$ |
| `Llama3-8B` (0-shot) | $-15.83_{\pm 0.41}$ | $-39.16_{\pm 0.43}$ | $86.38_{\pm 1.07}$ | $0.72_{\pm 0.10}$ | $0.76_{\pm 0.11}$ | $0.68_{\pm 0.14}$ |
| `Llama3-8B` (0-shot) + SIS | $-0.002_{\pm 0.001}$ | $-32.44_{\pm 0.15}$ | $91.58_{\pm 0.01}$ | $0.71_{\pm 0.11}$ | $0.75_{\pm 0.13}$ | $0.67_{\pm 0.15}$ |
| `Llama3-8B` (0-shot) + SMC | $-0.28_{\pm 0.24}$ | $-30.36_{\pm 0.68}$ | $92.78_{\pm 0.35}$ | $0.87_{\pm 0.06}$ | $0.87_{\pm 0.04}$ | $0.75_{\pm 0.07}$ |
| `Llama3-8B` (5-shot) | $-8.02_{\pm 4.62}$ | $-37.17_{\pm 0.21}$ | $86.77_{\pm 0.62}$ | $0.75_{\pm 0.10}$ | $0.79_{\pm 0.10}$ | $0.70_{\pm 0.13}$ |
| `Llama3-8B` (5-shot) + SIS | $-0.002_{\pm 0.0}$ | $-31.87_{\pm 0.42}$ | $92.24_{\pm 0.16}$ | $0.74_{\pm 0.10}$ | $0.79_{\pm 0.11}$ | $0.69_{\pm 0.14}$ |
| `Llama3-8B` (5-shot) + SMC | $-0.001_{\pm 0.002}$ | $-29.96_{\pm 0.24}$ | $93.05_{\pm 0.28}$ | $0.88_{\pm 0.06}$ | $0.87_{\pm 0.04}$ | $0.75_{\pm 0.07}$ |

Table 3: Full table of results. Please refer to §5.4 for plots and discussion.

## C Notation Glossary

| symbol | meaning |
|---|---|
| $N$ | string length (number of tokens) |
| EOS | end-of-string marker |
| $y, y' \in \Sigma$ | symbols in the alphabet $\Sigma$ |
| $\boldsymbol{y} \in \Sigma^*$ | string from the set of string $\Sigma^*$ |
| $\boldsymbol{y}_n \in \Sigma$ | $n^{\text{th}}$ symbol in string $\boldsymbol{y}$ |
| $\Sigma$ | alphabet; set of tokens |
| $\Sigma^*$ | set of all strings made from symbols in $\Sigma$ |
| $L$ | string length (number of words) |
| $x$ | word, defined as a sequence of tokens and identified as a single word by spaCy |
| $\mathbf{t}$ | target syntax tree |
| $P(\boldsymbol{y} \mid \mathbf{t})$ | posterior over strings |
| $P(\boldsymbol{y})$ | posterior over strings; abbreviation for $P(\boldsymbol{y} \mid \mathbf{t})$ when $\mathbf{t}$ is clear from context |
| $p(\boldsymbol{y})$ | language model prior of strings |
| $\varphi(\mathbf{t} \mid \boldsymbol{y})$ | likelihood; the probability that $\boldsymbol{y}$ has the syntax tree $\mathbf{t}$ |
| $\varphi(\boldsymbol{y})$ | likelihood; abbreviation for $\varphi(\mathbf{t} \mid \boldsymbol{y})$ when $\mathbf{t}$ is clear from context |
| $Z(\mathbf{t})$ | posterior normalization constant |
| $Z$ | posterior normalization constant; abbreviation for $Z(\mathbf{t})$ when $\mathbf{t}$ is clear from context |
| $M$ | sample size |
| $Q(\boldsymbol{y})$ | proposal distribution |
| $\psi(\boldsymbol{y})$ | shaping function; autoregressive tetratagger |
| $\widehat{P}(\boldsymbol{y})$ | posterior approximation |
| $\widehat{Z}$ | estimated normalization constant |
| $w$ | importance weight |
| $q(\boldsymbol{y} \mid z_n, z_{n+1})$ | bigram proposal |
| $z_n$ | part-of-speech tag for the $n^{\text{th}}$ word of $\mathbf{t}$ |
| $\mathcal{T}$ | set of tags (tetratagger) |
| $\mathbf{h}(x_\ell \mid \boldsymbol{y}) \in \mathbb{R}^d$ | transformer language model hidden state with dimensionality $d$ |
| $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{\mathcal{T} \times d}$ | tetratagger parameters |

Table 4: Notation glossary