TAGGEN: Enforcing Syntactic Structures with Tag-Based Control

Anonymous ACL submission

Abstract

Instruction fine-tuning has made it easier to control language model outputs for tasks like generating text with a specific topic or sentiment. By using targeted prompts with instructions and examples, instruction-tuned models can generally meet these control objectives. However, controlling the syntactic structure of the generated text remains a challenge. To address this issue, we introduce TAGGEN, a novel controlled decoding algorithm based on the parsing-as-tagging framework. TAGGEN uses syntactic taggers to guide the generation process, ensuring that each token aligns with the desired syntactic structure as it is generated. Our experiments with `GPT2` and `Llama3-8B` models show that TAGGEN significantly improves syntactic accuracy, increasing the F1 score from 8.34 to 34.63 for `GPT2-large` and from 36.86 to 47.29 for `Llama3-8B`. These results highlight the complexity of syntactic control and the effectiveness of TAGGEN in controlling the syntactic structure of the text, offering a promising approach for applications where precise control over syntax is essential.

1 Introduction

Language models (LMs) are widely deployed to generate text in natural language, e.g., in downstream tasks such as dialogue generation, summarization, and translation. However, to ensure a successful deployment of language models as text generators for domain-specific use cases, we need to know how to effectively and reliably control them, an area of research called controlled generation. There are many techniques in the literature that perform controlled generation. One line of work focuses on developing new algorithms to steer the generations to satisfy control targets at inference time (Krause et al., 2021; Liu et al., 2021; Schick et al., 2021, *inter alia*). Such methods have been successfully applied in various tasks, such as controlling generations' topics, sentiments, or toxicity.

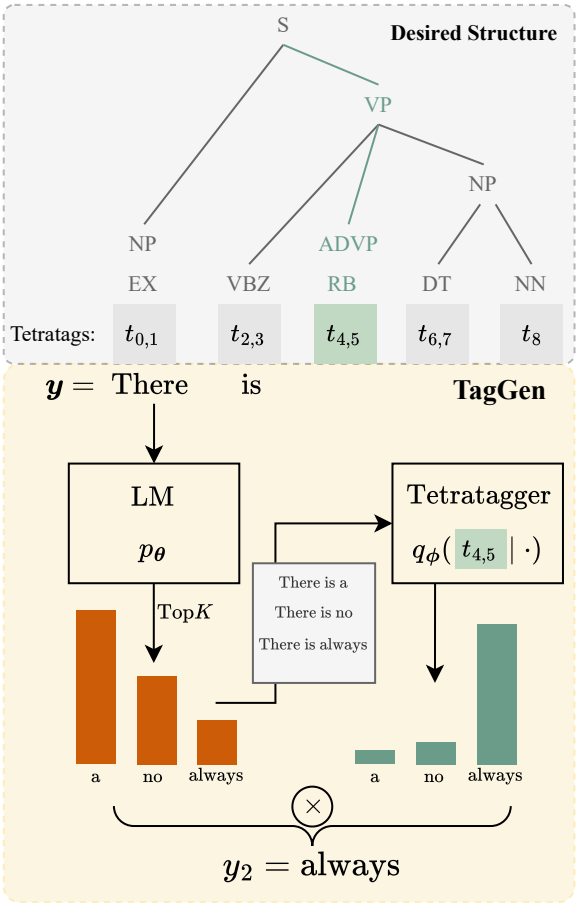

Figure 1: Overview of TAGGEN algorithm. At each generation step, we select the top K candidates from the language model, score them using Tetratagger, reweight their probabilities based on tag correctness, and sample from the adjusted distribution.

Recent advancements in instruction fine-tuning methods have made controlled decoding algorithms less appealing. This is because recent studies (Sun et al., 2023; Ashok and Poczos, 2024) show that simply prompting instruction fine-tuned LMs to follow a specific control target—such as generating text on a particular topic or with a positive sentiment—is typically sufficient to achieve the desired output, without the need for complex decoding algorithms.

A notable exception is syntactic control, where simple prompting fails to guide models to produce text with the desired syntactic structure (Sun et al., 2023; Ashok and Poczos, 2024). However, in certain cases, controlling the syntactic structure of sentences is crucial, e.g., in tasks that require stylistic consistency, such as legal or formal writing. Additionally, in psycholinguistics, syntactic control would allow for the automatic generation of stimuli to perform controlled experiments to study how syntax impacts cognitive load, memory, or reading time (Britton et al., 1982; Sandra H. Vos and Friederici, 2001). Finally, from an interpretability perspective, syntactic control provides a framework for understanding how syntax affects language models' behavior (Linzen et al., 2016; Futrell et al., 2019).

It is perhaps unsurprising that prompting alone is insufficient for reliably controlling syntax. Enforcing a global structure upon the generated string is inherently challenging. Most modern models are autoregressive, generating text token-by-token based on the previous context. To steer this process algorithmically, we would need a parser capable of estimating the likelihood that a given token will result in a syntactically correct sequence. Given the complexity of this task, in this paper, we explore whether controlled decoding techniques can improve the syntactic controllability of language models.

To improve syntactically controlled generation, in this paper, we propose a controlled decoding algorithm based on the parsing-as-tagging framework (Gómez-Rodríguez and Vilares, 2018; Vacareanu et al., 2020; Amini and Cotterell, 2022, *inter alia*). Therefore, we name our algorithm TAGGEN. 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 parses constituency trees by assigning tags to each word. To enforce a target constituency tree, we first convert it into a sequence of Tetratags. During generation, we multiply the next token probabilities given by the language model by the probability that the correct syntactic tag is assigned by Tetratagger, normalize the weights, and then sample the next token. This process is illustrated through an example in Fig. 1.

Our experiments with `GPT4` (OpenAI et al., 2024) align with previous findings, showing that the `GPT4` model has difficulty generating sentences with the desired constituency trees in zero- or few-shot settings. We then apply TAGGEN to `GPT2` models of different sizes (Radford et al., 2019), as well as to `Llama3-8B` models (Dubey et al., 2024). Sentences generated using TAGGEN adhere to the target syntax much better than those generated without it. For instance, TAGGEN improves the F1 score of `GPT2-large` generations from 8.34 to 34.63. Additionally, TAGGEN can be applied to instruction fine-tuned models like `Llama3-8B`, boosting its F1 score from 36.86 to 47.29. This performance is competitive with the 5-shot results of `GPT4` with an F1 score of 49.79, despite `Llama3-8B` being significantly smaller in size.

The rest of the paper is structured as follows: in §2 and §3, we explain the main building blocks of TAGGEN, namely weighted decoding and Tetratagging. Putting it together, §4 outlines the TAGGEN algorithm in detail. In §5, we explain the process of constructing the controlled generation dataset. §6 explores the task complexity by assessing the performance of `GPT4` model in solving the task. Finally, §7 presents the main experimental results.

2 Weighted Decoding

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

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}) \stackrel{\text{def}}{=} p_{\boldsymbol{\theta}}(\text{EOS} \mid \boldsymbol{y}) \prod_{t=1}^{|\boldsymbol{y}|} p_{\boldsymbol{\theta}}(y_t \mid \boldsymbol{y}_{<t}), \quad (1)$$

where $\text{EOS} \notin \Sigma$ is a distinguished, end-of-sequence token and $\boldsymbol{y}_{<t}$ denotes the length-$(t-1)$ prefix of $\boldsymbol{y}$.

Assume we want the strings decoded from the language model to have an attribute $\mathbf{a}$, e.g., a specific syntactic structure or topic. Therefore, we are interested in sampling from the following conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \mathbf{a})$. According to Bayes' rule, we can write

$$p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \mathbf{a}) = \frac{1}{Z} p_{\boldsymbol{\theta}}(\boldsymbol{y})\, p(\mathbf{a} \mid \boldsymbol{y}), \quad (2)$$

where $Z = \sum_{\boldsymbol{y} \in \Sigma^*} p_{\boldsymbol{\theta}}(\boldsymbol{y})\, p_{\boldsymbol{\theta}}(\mathbf{a} \mid \boldsymbol{y})$, and is intractable to compute since it involves summing over all the strings in Σ^*. Weighted decoding (Ghazvininejad et al., 2017; Holtzman et al., 2018;

Yang and Klein, 2021) is a heuristic algorithm to sample from Eq. (2). First, we assume that we have access to a controller $q_{\boldsymbol{\phi}}(\mathbf{a} \mid \boldsymbol{y})$, parameterized by $\boldsymbol{\phi}$ that models $p(\mathbf{a} \mid \boldsymbol{y})$. Given such a controller, the weighted decoding algorithm is outlined in Alg. 1.

The algorithm proceeds as follows: first, we start the decoded sequence with an empty string.[1] We take K_1 tokens that have the highest probability under $p_{\boldsymbol{\theta}}$. We then concatenate each with the current decoded string and compute the probability of the attribute $\mathbf{a}$ under the controller. We weigh the LM probabilities with the controller probabilities and adjust the importance of the controller probabilities with a hyperparameter λ. Finally, take the top K_2 tokens, sample the next token from normalized weights of the K_2 candidates, and append that to the current decoded string.

Algorithm 1 Weighted Decoding Algorithm

```
procedure WD(p_θ, q_ϕ, T)                    ▷ T max sequence length
    y = ε                                    ▷ start an empty string
    while |y| < T :
        y_1, ..., y_{K_1} = Top K_1(p_θ(· | y))
        for k ∈ {1, ..., K_1} :
            w_k = log p_θ(y_k | y) + λ log q_ϕ(a | y_k ∘ y)
        y_1, ..., y_{K_2} = Top K_2(w)
        y ~ Categorical(softmax(⟨w_1, ... w_{K_2}⟩))
        if y = EOS : return y
        y = y ∘ y                            ▷ append the sampled token
    return y
```

3 Tetratagger

A tetragger $q_{\boldsymbol{\phi}}$ is a probabilistic constituency parser, parameterized by $\boldsymbol{\phi}$, that works by assigning tags to words in the sentence. We show the set of all possible tags with $\mathcal{T}$, which consists of 4 unique tag types,[2] hence the name *tetra*tagger. Given a string of N words $\boldsymbol{y} = w_1 w_2 \cdots w_N$, its corresponding binarized constituency contains $2N-1$ nodes. Therefore, $q_{\boldsymbol{\phi}}$ models two probability distributions over tags per each word, except the last word to which we only assign 1 tag. Given $\boldsymbol{y}$, the probability of a tag sequence $\mathbf{t} = t_1 t_2 \cdots t_{2N-1}, t_n \in \mathcal{T}$ under $q_{\boldsymbol{\phi}}$ is

$$q_{\boldsymbol{\phi}}(\mathbf{t} \mid \boldsymbol{y}) \stackrel{\text{def}}{=} \prod_{n=1}^{N} q_{\boldsymbol{\phi}}(t_{2n-1} \mid \boldsymbol{y}) \prod_{n=1}^{N-1} q_{\boldsymbol{\phi}}(t_{2n} \mid \boldsymbol{y}). \quad (3)$$

[1] In practice, for instruction-tuned models, we start with the instruction, and for non-instruction-tuned models, we start the string with a distinguished beginning of the sequence token, i.e., BOS $\notin \Sigma$.

[2] 2 of the tag types encode information about the leaf nodes in the constituency tree, and the other 2 encode information about the internal nodes.

To parse a string $\boldsymbol{y}$ to a constituency tree, first, we find the valid tag sequence with the highest probability under $q_{\boldsymbol{\phi}}$ using a linear-time algorithm (§3.4; Kitaev and Klein, 2020). Second, another linear-time deterministic algorithm (§3.3; Kitaev and Klein, 2020) transforms the tag sequence into the constituency tree.

3.1 Modeling

Tetratagger is modeled by a language model and 2 projection matrices. Concretely, to obtain $q_{\boldsymbol{\phi}}(t_{2n-1} \mid \boldsymbol{y})$ and $q_{\boldsymbol{\phi}}(t_{2n} \mid \boldsymbol{y})$ in Eq. (3), we first pass $\boldsymbol{y}$ to a language model and extract the representation of w_n from the last layer of the language model, i.e., $\mathbf{h}(w_n \mid \boldsymbol{y}) \in \mathbb{R}^d$. Next, we apply 2 independent projection matrices, $P_1, P_2 \in \mathbb{R}^{|\mathcal{T}| \times d}$, to this representation. Finally, we apply softmax to predict the distribution over the tags:

$$q_{\boldsymbol{\phi}}(t_{2n-1} \mid \boldsymbol{y}) \stackrel{\text{def}}{=} \text{softmax}\big(P_1 \cdot \mathbf{h}(w_n \mid \boldsymbol{y})\big) \quad (4)$$

$$q_{\boldsymbol{\phi}}(t_{2n} \mid \boldsymbol{y}) \stackrel{\text{def}}{=} \text{softmax}\big(P_2 \cdot \mathbf{h}(w_n \mid \boldsymbol{y})\big) \quad (5)$$

Given a dataset of strings and their corresponding ground-truth Tetratags, we learn $\boldsymbol{\phi}$ by maximizing the ground-truth tag likelihood given the strings.

Tokens vs. Words. While the language model in Eq. (4) is assumed to represent words in a string, language models operate over *tokens* and not words. Therefore, following Kitaev and Klein (2020), we set $\mathbf{h}(w_n \mid \boldsymbol{y})$ to be the representation of the *last token* of w_n extracted from the last layer of the language model.

Autoregressive Language Models. Originally Kitaev and Klein (2020) used BERT (Devlin et al., 2019) as the language model in tetratagger. The representation of w_n extracted from BERT includes bidirectional context, meaning it incorporates information from both the words preceding and following the nth word. In contrast, state-of-the-art language models are autoregressive, meaning the representation of w_n is based only on its preceding words. While one might argue that bidirectional information is crucial for accurately predicting syntactic structure, we achieve a notable F1-score of 70.32 using only autoregressive models. For more details on the tagger's accuracy, refer to §7.1.

4 TagGen

Having covered weighted decoding and Tetratagger, we are now ready to explain TagGen through

an example. TAGGEN is a weighted decoding algorithm for generating text from language models given a syntactic structure $\mathbf{a}$. Assume we want the generated text to have a constituency tree structure $\mathbf{a}$, visualized in Fig. 1. First, we transform the constituency tree to a sequence of tags $\mathbf{t} = t_0, t_1, \cdots, t_8$ using a linear-time deterministic algorithm (§3.1; Kitaev and Klein, 2020). Next, assuming that the Tetratagger is using an autoregressive LM backbone, we can write the Tetratag probability in Eq. (3) as

$$q_{\boldsymbol{\phi}}(\mathbf{t} \mid \boldsymbol{y}) = \prod_{n=1}^{N} q_{\boldsymbol{\phi}}(t_{2n-1} \mid \boldsymbol{y}_{\leq n}) \prod_{n=1}^{N-1} q_{\boldsymbol{\phi}}(t_{2n} \mid \boldsymbol{y}_{\leq n}). \quad (6)$$

Conveniently, the desired syntactic structure $\mathbf{a}$ is now factorized into each token in the sequence. We then adjust the weighted decoding algorithm Line 6 in Alg. 1 to use Tetratagger Eq. (6) as the controller. Concretely, assume that we already decoded the string $\boldsymbol{y}$ = "There is" of length 2. We look at the top K_1 tokens from the language model. In the example in Fig. 1, the top 3 tokens are a, no, and always. We then pass the 3 candidates: "There is a", "There is no", and "There is always" to the Tetratagger and compute the probability of the ground-truth Tetratags assigned to the last word, i.e., t_3, t_4. The ground-truth tags t_3 and t_4 describe how this word is placed in the syntactic structure, i.e., an adverb that is a part of a verb phrase. Finally, we compute the weights for the kth candidate as the sum of the probability of the candidate under the language model and the probability of the ground-truth tags given the candidate, i.e.,

$$w_k = p_{\boldsymbol{\theta}}(y_k \mid \boldsymbol{y}) + \lambda q_{\boldsymbol{\phi}}(t_3 \mid y_k \circ \boldsymbol{y}) + \lambda q_{\boldsymbol{\phi}}(t_4 \mid y_k \circ \boldsymbol{y}). \quad (7)$$

After weighing the candidates with Tetratagger, we sample from the top K_2 candidates and proceed with the algorithm Alg. 1. In our example, the best candidate after reweighting is "There is always", which matches the desired constituency structure.

Word Boundaries. We again highlight the fact that Tetratagger assigns tag probabilities to the last token of each word. Therefore, if a candidate token y_k is not the last token of the word, we do not pass $y_k \circ \boldsymbol{y}$ to the Tetratagger and automatically assign weight 1 to it. 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.

	$\mathcal{D}_1$	$\mathcal{D}_2$
Number of trees	116	200
Mean height	5.90	7.75
Max height	10.00	18.00
Mean leaf nodes	6.70	9.49
Max leaf nodes	12.00	20.00
Mean tree size	18.71	27.46

Table 1: Statistics for datasets $\mathcal{D}_1$ and $\mathcal{D}_2$. $\mathcal{D}_2$ includes trees that are larger on average compared to $\mathcal{D}_1$.

5 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. We produce two different datasets, $\mathcal{D}_1$ and $\mathcal{D}_2$, of varying complexity. $\mathcal{D}_1$ consists of a total of 116 trees with the most frequent non-terminal labels in our dataset,[3] while $\mathcal{D}_2$ consists of 200 more complex trees. Tab. 1 provides the dataset statistics. In both datasets, the minimum height of the tree is 3, the minimum number of leaf nodes is 2 and the minimum tree size is 5.

6 Task Complexity

Before evaluating TAGGEN, 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.

After initial experiments with several prompt formulations, we use the best-performing prompt in Fig. 2 to instruct `GPT4-Turbo-preview` to generate sentences that follow a specific syntactic

[3]The most frequent non-terminal labels are `NP, VP, S, NN, ., PRP, DT, IN, PP, VB, RB, JJ, VBP, NNS, VBZ, VBD`.

Metrics	$\mathcal{D}_1$			$\mathcal{D}_2$		
	0-shot	**5-shot**	**1-shot (gold)**	**0-shot**	**5-shot**	**1-shot (gold)**
% correct sentence length	55.17	50.86	61.21	34.67	41.21	59.80
% exact tree match	22.41	27.59	30.17	15.09	15.58	33.67
% exact tree structure match	33.62	35.34	42.24	17.09	21.10	46.73
Tag Accuracy	83.95	91.81	91.41	86.07	86.27	94.57
BLEU	57.65	65.39	70.24	63.78	68.36	79.41
F1	53.96	60.42	68.84	43.55	49.79	71.05

Table 2: Results of GPT4 with 0-shots, 5-shots and 1-shot of gold example on $\mathcal{D}_1$ and $\mathcal{D}_2$ datasets. We observe GPT4 does not manage to generate sentences with the correct syntactic structure in most cases, and achieves F1 score of only 60.42 and 49.79 on $\mathcal{D}_1$ and $\mathcal{D}_2$ with 5 examples. Even when the model has access to the ground-truth sentence (1-shot (gold)) it only achieves F1 score of 68.84 and 71.05 on the two datasets.

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:
Syntax tree: {SYNTAX}
Answer: {SENTENCE}
----------------------------------------
Now generate a sentence for the following
parse tree:
Syntax tree: {SYNTAX}
Answer:
```

Figure 2: The prompt used for generating text with GPT4-Turbo-preview model.

structure. The prompt text is adjusted for the other prompt techniques, 0-shot and 5-shot, to match the requirements of those setups by either providing more examples (5-shot) or removing examples (0-shot). To include the constituency trees in this prompt, we need to linearize the tree. This is done using the bracketing representation of the trees. We further remove the part-of-speech tags, and the words, and replace them with question marks. For example, the linearized representation of the desired tree depicted in Fig. 1 is `(S (NP (EX ?)) (VP (VBZ ?) (ADVP (RB ?)) (NP (DT ?) (NN ?))))`. The GPT4-generated sentences are then post-processed to include only the requested sentence. Then, they are parsed to constituency trees and Tetratags to evaluate the correctness of trees and tag sequences respectively.

We evaluate the following metrics:

- **% correct sentence length** is the percentage of generated sentences that match the exact word count specified by the constituent tree.
- **% exact tree match** is the percentage of sentences whose generated parse trees match the desired parse trees exactly, both in structure and labels.
- **% exact tree structure match** is the percentage of sentences whose generated parse trees match the desired parse trees in structure (ignoring labels).
- **Tag accuracy** is the accuracy of generating sentences with the desired tag sequence, calculated only for a subset of generated sentences where the tag sequence corresponding to the generated sentence and the desired tag sequence are of equal length.
- **BLEU** is the mean BLEU score between the tag sequences of the desired constituency trees and those of the generated sentences, measuring n-gram overlap and applying a brevity penalty to account for length differences.
- **F1** is the mean bracketing F1 score comparing the desired constituency trees with the constituency trees of the generated sentences.

The results are reported in Tab. 2. First, we observe that only 55.17% and 41.21% of GPT4 generations have the desired length on $\mathcal{D}_1$ and $\mathcal{D}_2$, 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.

Models	Internal Acc.	Leaf Acc.	F1
GPT2-small	79.63	92.49	65.14
GPT2-medium	80.64	93.08	67.50
GPT2-large	80.81	93.45	67.91
Llama3-8B	82.18	94.19	70.32

Table 3: Fine-tuning different Tetra-Taggers. **Internal Acc.**: Tag accuracy of internal nodes, **Leaf Acc.**: Tag accuracy of leaf nodes, **F1**: Bracketing F1-score.

In all experimental setups, as reflected in Tab. 2, GPT4 struggles to generate sentences with the desired structure. On the easier dataset $\mathcal{D}_1$, we achieve F1 scores of only 53.96 and 60.42 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 1-shot (gold) setting. Surprisingly, even with the gold sentences included in the prompt, the model fails to always copy the sentence from the prompt into the output and only reaches an F1 score of 68.42 and 71.05 on $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. Overall the results suggest that the ability to generate sentences with a specific syntax is challenging even for state-of-the-art language models.

7 Controlled Generation with TAGGEN

We experiment with models of different sizes: GPT2-small (124M parameters), GPT2-medium (355M parameters), GPT2-large (774M parameters), and Llama3-Instruct (8B parameters). In all our experiments, we used $K_1 = 30$, $K_2 = 10$ and $\lambda \in \{0, 1\}$ (See §7.3, §7.4). Notably, setting $\lambda = 0$ indicates that TAGGEN is not applied, and the language model operates without control until producing the number of requested words. For each autoregressive language model, we used the corresponding Tetratagger model trained with the same LM as the backbone.

7.1 Training Tetrataggers

We train Tetrataggers 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 that we use for generating text. This is essential to align the tokenization scheme of Tetratagger with the language model we use to generate text. We train 4 different versions of Tetratagger depending on the LM backbone. For efficiency reasons, to train a Tetratagger with Llama3-8B as the LM backbone, we applied 4-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. We trained all Tetratagger models for 4 epochs on the Penn Treebank dataset (Marcus et al., 1993), using the cross-entropy loss.

In Tab. 3, we report how accurately Tetratagger can predict correct tags for the leaf nodes (Leaf Acc.) and for the internal nodes (Internal Acc.). Furthermore, we convert the sequence of predicted tags to constituency trees and compute the bracketing F1 score. Our best-performing Tetratagger with Llama3-8B backbone achieves a high F1 score of 70.32 even though it does not use any bidirectional information and does not have access to the whole sequence when predicting each tag.

7.2 Controlled Generation Results

We test our decoding algorithm TAGGEN with models of varying sizes. For GPT2 family of models, our prompt only includes the beginning-of-sequence (BOS) token, and for Llama3-8B models, we use a similar prompt to the one used for GPT4 model; see App. B. We repeat each experiment 10 times for GPT2 and 3 times for Llama3-8B models. The mean and standard deviation of evaluation metrics are reported in Tab. 4. We refer the reader to App. D for additional experimental results.

First, we observe a significant increase in tag accuracy, BLEU score, and F1 score when we apply TAGGEN to the GPT2 family of models. Prior to applying our controlled generation algorithm, GPT2 outputs did not match the desired structure, as reflected in very low F1 scores. This is expected, as the models are oblivious to the desired structure. Upon applying TAGGEN, however, we observe a significant boost in all metrics. For example, applying TAGGEN on GPT2-large model makes a 4× improvement in F1 scores. Moreover, we observe that larger GPT2 models achieve higher F1 scores when using TAGGEN algorithm.

Second, we observe that TAGGEN can significantly improve the instruction-tuned Llama3-8B model both in 0-shot and in 5-shot settings. For example, TAGGEN increases the F1 score of Llama3-8B (5-shot) model on $\mathcal{D}_1$ from 56.10 to 62.37 and from 36.86 to 47.29 on $\mathcal{D}_2$ making it competitive to 5-shot performance of GPT4

Models	$\mathcal{D}_1$			$\mathcal{D}_2$		
	Tag Accuracy	BLEU	F1	Tag Accuracy	BLEU	F1
GPT2-small	37.23±1.61	8.28±1.03	9.4±1.24	34.85±0.49	14.62±1.35	7.16±0.40
GPT2-small + TAGGEN	72.80±0.67	41.02±1.72	36.82±2.19	68.88±0.99	42.90±1.76	31.03±1.63
GPT2-medium	35.50±1.44	10.17±16.8	11.41±1.23	33.50±0.53	15.51±1.75	8.82±0.57
GPT2-medium + TAGGEN	73.1±1.00	44.44±2.71	39.09±1.77	69.61±1.22	45.53±2.12	34.33±1.55
GPT2-large	36.24±1.06	9.58±0.68	10.51±1.13	33.64±0.95	14.46±1.14	8.35±0.66
GPT2-large + TAGGEN	**73.83**±1.02	**46.2**±2.32	40.40±1.86	**70.79**±0.94	**46.29**±1.55	34.63±1.58
Llama3-8B (0-shot)	52.57±1.47	28.18±0.41	37.52±1.42	43.22±0.54	27.77±1.21	23.92±0.58
Llama3-8B (0-shot) + TAGGEN	67.63±1.29	35.30±2.5	52.14±0.47	62.65±0.45	36.34±1.59	38.69±0.93
Llama3-8B (5-shot)	61.62±0.84	30.75±0.83	56.10±2.25	51.22±1.75	33.85±0.52	36.86±1.27
Llama3-8B (5-shot) + TAGGEN	73.42±0.95	42.30±2.10	**62.37**±0.78	68.51±0.58	43.86±0.17	**47.29**±2.84

Table 4: Results on controlling the syntactic structure of generations. TAGGEN increases adherence to the desired syntax across all methods and metrics. Applying TAGGEN to Llama3-8B (5-shot) increases the F1 score from 36.86 to 47.29, which is competitive to GPT4 5-shot F1 score of 49.79.

model. This is particularly promising, given that Llama3-8B is significantly smaller than GPT4 in number of parameters.

7.3 Perplexity Trade-offs

In this section, we explore the trade-off between the quality of the generated sentences by TAGGEN and their alignment with the desired syntax tree by varying the value of λ. Increasing λ enhances the influence of Tetratagger by assigning more weight to q_ϕ probability, thereby improving tree-structural accuracy. However, we observed that higher values of λ can lead to lower-quality sentences, as Tetratagger potentially introduces noise into the generated outputs. To better understand this balance, we measured perplexity—an indicator of text quality—alongside the bracketing F1 score of the generated trees.

Fig. 3 illustrates the trade-off between perplexity and bracketing F1 score across different GPT2 models. The mean perplexity was calculated over 5 runs of each model on $\mathcal{D}_1$ dataset using Llama3-8B model. As λ increases, we observe that the F1 score improves initially but plateaus for $\lambda \geq 1.5$, while perplexity continues to rise, indicating a degradation in sentence quality. Both $\lambda = 0.5$ and $\lambda = 1$ accomplish a balance between perplexity and bracketing F1 score, with $\lambda = 1$ achieving better tree-structured sentences but of lower quality. For our experiments, we chose $\lambda = 1$, slightly sacrificing sentence quality in favor of improved syntactic alignment. As expected, larger models tend to produce higher-quality sentences, therefore, using larger models results in improved sentence generation.

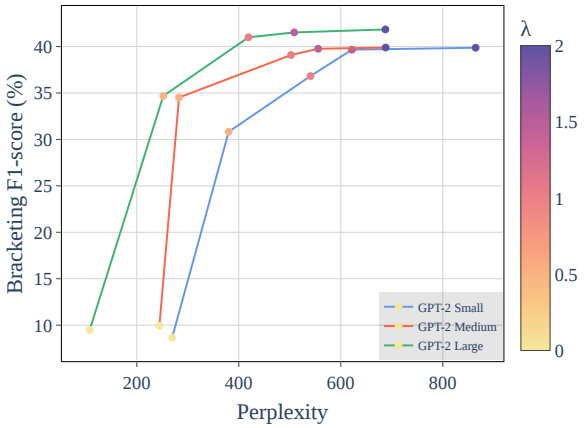

Figure 3: Relationship between perplexity and bracketing F1-score for various λ values on $\mathcal{D}_1$ dataset. In general, increasing λ improves F1 score, but hurts fluency by increasing perplexity. The improvement on F1 score plateaus after $\lambda \geq 1.5$.

7.4 Decoding time

In this section, we analyze the computational cost of TAGGEN. Concretely, as outlined in Alg. 1, by increasing the number of candidates we consider at each generation step, i.e. increasing K_1, we are essentially increasing the decoding time. This is due to the need to pass all the candidates to the Tetratagger.[4] At the same time, increasing K_1 can lead to a better bracketing F1 score. Therefore, given the computational resources at hand, K_1 is

[4]For small-enough K_1 values, decoding time will not be affected since all the candidates can be processed in parallel by Tetratagger, and only throughput will decrease.

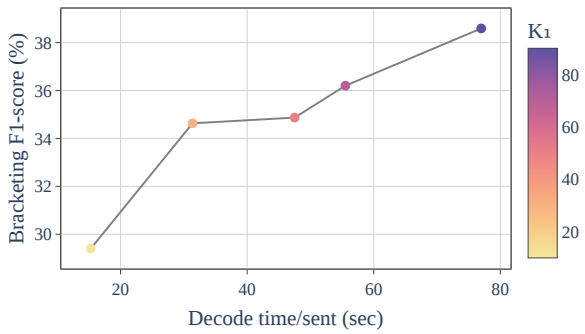

Figure 4: Relationship between decode time/sent (sec) and bracketing F1 score (%) for various K_1 values on $\mathcal{D}_2$ dataset using `GPT2-large` model. As K_1 increases, the decoding time of our algorithm rises, but the F1 score also improves, leading to better alignment of generated sentences with the syntactic trees.

the knob to tune to find the ideal value that balances decoding time and F1 score.

Fig. 4 shows the relationship between decoding time per sentence (in seconds) and the corresponding F1 score by varying values of K_1 for the `GPT2-large` model. The experiments were conducted on a single `NVIDIA GeForce RTX 4090 GPU` with a batch size of 8. We utilized $\mathcal{D}_2$ dataset, which contains larger syntax trees on average as explained in 1. Consequently, our algorithm requires more time to process these trees effectively. As K_1 increases, there is a noticeable improvement in the F1 score from approximately 30% to nearly 38%, however, this comes at the cost of increased decoding time. In our final results in 4, we used $K_1 = 30$, which provides a reasonable balance between decoding time and F1 score.

8 Conclusion

In this paper, we introduced TAGGEN, a decoding algorithm designed to control the syntactic structure of text generated by language models. TAGGEN leverages a parsing-as-tagging framework, guiding the generation process by incorporating syntactic taggers. At each step of text generation, the algorithm computes the product of two distributions: the probability of the next token as given by the language model and the probability that the token adheres to the desired syntactic structure, as determined by the tagger. The next token is then sampled from the normalized product of these two distributions, ensuring that both linguistic content and syntactic correctness are taken into account. Our experiments with `GPT2` and `Llama3-8B` models demonstrate that TAGGEN achieves significant improvements in generating text that aligns with target syntactic structures. These results suggest that TAGGEN offers a promising approach for enhancing the syntactic controllability of language models.

Limitations

Pruned-Height Trees. TAGGEN 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 first K_1 tokens 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 recognize that our approach may introduce unexpected biases or inaccuracies, potentially resulting 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 TAGGEN, 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 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8884–8900, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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 MJ Penland. 1982. Effects of text structure on use of cognitive capacity during reading. *Journal of Educational Psychology*, 74(1):51.

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019. Controllable paraphrase generation with a syntactic exemplar. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5972–5984, Florence, Italy. 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. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan

Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 32–42, Minneapolis, Minnesota. Association for Computational Linguistics.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. 2017. Hafez: an interactive poetry generation system. In *Proceedings of ACL 2017, System Demonstrations*, pages 43–48, Vancouver, Canada. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324, Brussels, Belgium. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Maxwell Forbes, Antoine Bosselut, David Golub, and Yejin Choi. 2018. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1638–1649, Melbourne, Australia. Association for Computational Linguistics.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength Natural Language Processing in Python.

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*.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686, Melbourne, Australia. Association for Computational Linguistics.

Nikita Kitaev and Dan Klein. 2020. Tetra-tagging: Word-synchronous parsing with linear-time inference.

In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6255–6261, Online. 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: EMNLP 2021*, pages 4929–4952, Punta Cana, Dominican Republic. Association for Computational Linguistics.

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*, 4:521–535.

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 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6691–6706, Online. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

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

Herbert Schriefers Sandra H. Vos, Thomas C. Gunter

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):65–103.

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*, 9:1408–1424.

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 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3155–3168, Singapore. Association for Computational Linguistics.

Robert Vacareanu, George Caique Gouveia Barbosa, Marco A. Valenzuela-Escárcega, and Mihai Surdeanu. 2020. Parsing as tagging. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5225–5231, Marseille, France. European Language Resources Association.

John Wieting and Kevin Gimpel. 2018. ParaNMT-50M: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462, Melbourne, Australia. 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 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Kevin Yang and Dan Klein. 2021. 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*, pages 3511–3535, Online. Association for Computational Linguistics.

A Experimental Details

We trained our 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. The dataset consists of syntactic trees representing English sentences:

- **Training set**: 39832 trees
- **Development set**: 1700 trees
- **Test set**: 2426 trees

Our tag vocabulary consists of 231 labels in total. For GPT4, we used the OpenAI API[5] 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, for training Tetratagger, we used https://huggingface.co/meta-llama/Meta-Llama-3-8B under license META LLAMA 3 COMMUNITY LICENSE AGREEMENT for Llama3-8B, https://huggingface.co/openai-community/gpt2 for GPT2-small, https://huggingface.co/openai-community/gpt2-medium for GPT2-medium and https://huggingface.co/openai-community/gpt2-large for GPT2-large under MIT License. We tested our decoding algorithm with all these GPT2 models and Llama3-Instruct (https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct).

All constituent trees utilized in our experiments were represented in the bracketing representation, which aligns with standard practices in syntactic parsing. Batch size was set to 8 for GPT2 models and 2 for Llama3-8B models due to memory limitations. All experiments were conducted on a single NVIDIA GeForce RTX 4090 GPU with 24 GiB memory.

B Llama3-8B Prompt

For Llama3-Instruct model, we slightly changed and adjusted the prompt in Fig 2 in the format provided by Dubey et al. (2024), incorporating the necessary special tokens and roles to achieve the best results. Because TAGGEN can produce the generated sentence directly, the final sentence of our prompt includes a quotation mark at the end, indicated by the role "assistant". The prompt text is adjusted for 0-shot and 5-shots techniques.

[5] https://openai.com/policies/terms-of-use/

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:
Syntax tree: {SYNTAX}
The generated sentence is: "{SENTENCE}"
----------------------------------------
Now generate a sentence for the following
parse tree:
Syntax tree: {SYNTAX}
----------------------------------------
The generated sentence is: "
```

Figure 5: The prompt used for generating text with Llama3-Instruct model. Note that the exact format of the prompt, including special tokens and roles, is not provided for clarity.

C Evaluation

For evaluating our generated sentences we used metrics that assess both the tag sequences (BLEU score and Tag Accuracy) and syntactic tree structures produced from these sentences (F1 score, % exact tree match and % exact tree structure match). For parsing the generated sentences into constituent trees, we used the Berkeley Neural Parser[6] through spaCy (Honnibal et al., 2020). The predicted syntactic trees were then compared to the ground truth syntax trees in our dataset. For generating the tag sequences from our produced sentences, we utilized the Tetratagger model trained with the respective model used in the decoding. For the results of GPT4, we employed the pre-trained Tetratagger model by Kitaev and Klein (2020). The tag sequences generated were compared to the respective ground-truth tag sequences, which were produced from the deterministc algorithm of Tetratagger (§3.1; Kitaev and Klein, 2020).

D Full Experimental Results

Here, we report our full experimental results, including more metrics. All the experiments were conducted with $K_1 = 30$, $K_2 = 10$ and $\lambda = \{0, 1\}$.

[6] https://pypi.org/project/benepar/

Metrics	without control			TAGGEN		
	gpt2-small	**gpt2-medium**	**gpt2-large**	**gpt2-small**	**gpt2-medium**	**gpt2-large**
			$\mathcal{D}_1$			
% exact tree match	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.30 ± 0.46	0.83 ± 0.87	0.66 ± 0.90
% exact tree structure match	20.35 ± 5.11	20.59 ± 2.63	17.46 ± 3.37	38.25 ± 3.98	41.50 ± 4.96	44.86 ± 4.01
Tag Accuracy	37.23 ± 1.61	35.50 ± 1.44	36.24 ± 1.06	72.80 ± 0.67	73.1 ± 1.00	73.83 ± 1.02
BLEU	8.28 ± 1.03	10.17 ± 1.68	9.58 ± 0.68	41.02 ± 1.72	44.44 ± 2.71	46.2 ± 2.32
F1	9.4 ± 1.24	11.41 ± 1.23	10.51 ± 1.13	36.82 ± 2.19	39.09 ± 1.77	40.40 ± 1.86
			$\mathcal{D}_2$			
% exact tree match	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	00.27 ± 00.45
% exact tree structure match	14.91 ± 2.84	15.46 ± 2.60	15.98 ± 1.4	26.00 ± 2.43	25.73 ± 2.46	29.61 ± 3.05
Tag Accuracy	34.85 ± 0.49	33.50 ± 0.53	33.64 ± 0.95	68.88 ± 0.99	69.61 ± 1.22	70.79 ± 0.94
BLEU	14.62 ± 1.35	15.51 ± 1.75	14.46 ± 1.14	42.90 ± 1.76	45.53 ± 2.12	46.29 ± 1.55
F1	7.16 ± 0.40	8.82 ± 0.57	8.35 ± 0.66	31.03 ± 1.63	34.33 ± 1.55	34.63 ± 1.58

Table 5: Results of `GPT2` models. Applying TAGGEN significantly improves all metrics.

Metrics	without control			TAGGEN	
	0-shot	**5-shots**	**1-shot (gold)**	**0-shot**	**5-shots**
			$\mathcal{D}_1$		
% exact tree match	4.44 ± 1.42	16.84 ± 1.99	37.04 ± 2.27	4.08 ± 1.20	10.13 ± 3.28
% exact tree structure match	32.36 ± 2.29	36.63 ± 4.71	57.77 ± 2.69	37.87 ± 3.05	46.99 ± 1.96
Tag Accuracy	52.57 ± 1.47	61.62 ± 0.84	74.71 ± 1.12	67.63 ± 1.29	73.42 ± 0.95
BLEU	28.18 ± 0.41	30.75 ± 0.83	46.77 ± 0.28	35.30 ± 2.5	42.30 ± 2.10
F1	37.52 ± 1.42	56.10 ± 2.25	77.73 ± 0.97	52.14 ± 0.47	62.37 ± 0.78
			$\mathcal{D}_2$		
% exact tree match	0.54 ± 0.44	5.58 ± 1.28	24.54 ± 3.81	0.38 ± 0.27	4.81 ± 0.3
% exact tree structure match	22.67 ± 2.86	26.79 ± 3.00	50.27 ± 1.57	21.94 ± 0.59	28.26 ± 2.78
Tag Accuracy	43.22 ± 0.54	51.22 ± 1.75	70.79 ± 1.46	62.65 ± 0.45	68.51 ± 0.58
BLEU	27.77 ± 1.21	33.85 ± 0.52	54.37 ± 1.17	36.34 ± 1.59	43.86 ± 0.17
F1	23.92 ± 0.58	36.86 ± 1.27	66.60 ± 1.98	38.69 ± 0.93	47.29 ± 2.84

Table 6: Results of `Llama3-Instruct` without control and with TAGGEN. Similar to `GPT4` results, the model fails to achieve perfect accuracy even when the ground-truth sentence is included in the prompt as an example. Applying TAGGEN improves adherence to the syntactic structure in both 0-shot and 5-shot settings.