Sketch-Guided Constrained Decoding for Boosting Blackbox Large Language Models without Logit Access

Saibo Geng, Berkay Döner, Chris Wendler, Martin Josifoski, Robert West
EPFL
{saibo.geng, berkay.doner, chris.wendler, martin.josifoski, robert.west}@epfl.ch

Abstract

Constrained decoding, a technique for enforcing constraints on language model outputs, offers a way to control text generation without retraining or architectural modifications. Its application is, however, typically restricted to models that give users access to next-token distributions (usually via softmax logits), which poses a limitation with blackbox large language models (LLMs). This paper introduces *sketch-guided constrained decoding* (SketchGCD), a novel approach to constrained decoding for blackbox LLMs, which operates without access to the logits of the blackbox LLM. SketchGCD utilizes a locally hosted auxiliary model to refine the output of an unconstrained blackbox LLM, effectively treating this initial output as a "sketch" for further elaboration. This approach is complementary to traditional logit-based techniques and enables the application of constrained decoding in settings where full model transparency is unavailable. We demonstrate the efficacy of SketchGCD through experiments in closed information extraction and constituency parsing, showing how it enhances the utility and flexibility of blackbox LLMs for complex NLP tasks.[1]

1 Introduction

Large language models (LLMs) have seen a remarkable expansion in scope, being used for diverse tasks including tool interaction, SQL translation, robotic navigation and item recommendations, where adherence to specific constraints is paramount (Bubeck et al., 2023; Schick et al., 2023; Poesia et al., 2022; Shah et al., 2022; Zhang et al., 2023; Hua et al., 2023). Despite their versatility, LLMs often struggle with constraint adherence in few-shot scenarios, leading to outputs that violate task-specific requirements (Chen and Wan, 2023; Agrawal et al., 2023; Huang et al., 2023).

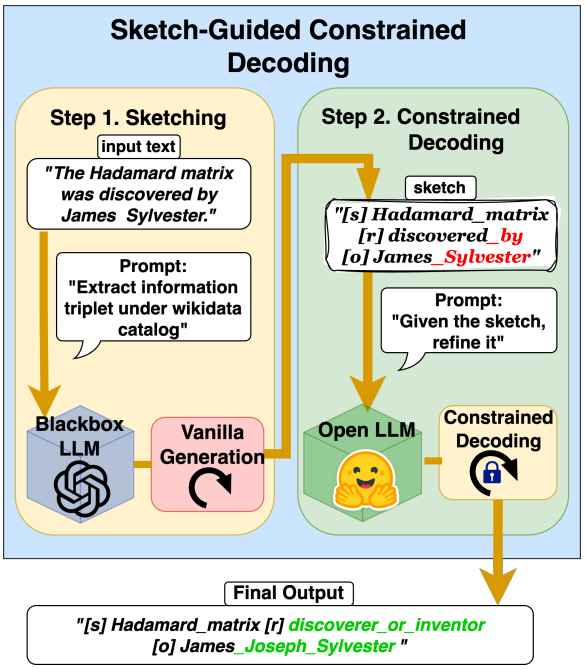


Figure 1: **Overview of sketch-guided constrained decoding (SketchGCD).** In the initial *sketching* phase, a blackbox LLM generates a preliminary "sketch" answer without applying any constraints. Then, in the *constrained decoding* phase, an auxiliary model, the constrained decoder, refines the sketch. The refined, final output respects the specified constraints by construction.

Constrained decoding offers a solution, restricting model outputs to respect predefined constraints without necessitating model retraining or architectural modifications (Poesia et al., 2022; Shin et al., 2021; Beurer-Kellner et al., 2023; Scholak et al., 2021; Geng et al., 2023). However, existing constrained decoding methods require access to the model's logits during inference, which is not always feasible in practice (cf. Appendix A). Since the most powerful LLMs tend to be commercial and blackbox (Lee et al., 2023), this has restricted the application of constrained decoding methods.

[1]Code and data available at https://github.com/epfl-dlab/SketchGCD

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 234–245

Contributions. To overcome this restriction, we present *sketch-guided constrained decoding* (SketchGCD), which bypasses the need for direct logit access. SketchGCD uses a locally hosted (lightweight) open-source LLM to refine the outputs of a (heavyweight) blackbox LLM to satisfy the specified constraints. We validate our method on closed information extraction, where the constraints require generating triples grounded in a knowledge base, and constituency parsing, where the constraints require generating tree-structured outputs. Our experiments show that SketchGCD significantly boosts the performance of LLMs and beats previous approaches by a wide margin.

2 Method

SketchGCD splits the constrained decoding task into two distinct phases: *sketching* and *constrained decoding*.

During *sketching,* a *sketcher*—a powerful blackbox LLM denoted as P_{sk}—is employed. It interprets an instruction I alongside a set of demonstration pairs $D = \{(x^i, y^i)\}_{i=1}^n$, producing a preliminary draft y^* via unconstrained decoding:

$$y^* \approx \arg\max_{y \in S} P_{\text{sk}}(y \mid I, D, x), \tag{1}$$

where S is the set of all possible sequences.

Constrained decoding is done by a *constrained decoder,* a smaller-scale, locally hosted LLM P_{cg}. Given an instruction I_{cg}, a set of input–sketch–output demonstrations $D_{\text{cg}} = \{(x^i, y^i, z^i)\}_{i=1}^n$, the original input x, and the sketch y^*, it refines y^* into

$$z^* \approx \arg\max_{z \in S \cap C} P_{\text{cg}}(z \mid I_{\text{cg}}, D_{\text{cg}}, x, y^*), \tag{2}$$

subject to constraints C. (Optionally, x and x^i may be omitted, with loss of information.)

The sketcher's output y^* is typically of high quality, encapsulating the necessary information for the constrained decoder to produce the final sequence z^* that adheres to the constraints C. Given the quality of y^*, the constrained decoder can be implemented using a much smaller model, as its primary task is to rewrite the sketch y^* with the help of constrained decoding, thus facilitating deployment on standard consumer-grade hardware.

On the contrary, classical, direct few-shot prompting with constrained decoding would usually require a larger constrained generator P_{cg} to be run locally, in order to find

$$w^* \approx \arg\max_{w \in S \cap C} P_{\text{cg}}(w \mid I, D, x). \tag{3}$$

Another basic alternative, unconstrained few-shot prompting (Brown et al., 2020), yields y^* as the end product.

SketchGCD builds on the expectation that the constrained refined output z^* should be at least as good as both y^* (as z^* respects the constraints) and w^* (as P_{sk} is a more powerful LLM than P_{cg}).

3 Experiments

In our experimental setup, we evaluate the efficacy of SketchGCD by comparing it against two established baselines: (1) few-shot-prompted unconstrained decoding with powerful blackbox LLMs (Eq. 1) and (2) few-shot-prompted constrained decoding with open-source LLMs (Eq. 3). The SketchGCD method remains flexible and is agnostic to the exact implementation of constrained decoding. Here we adopt the grammar constrained decoding framework of Geng et al. (2023), but any other constraining method can be plugged in.

In our evaluation, we distinguish between sequences that are *valid* (i.e., that satisfy the constraints) and those that are *correct* (i.e., those that are equal to the intended output for the given input). A valid output is a prerequisite for being correct, but it is not the sole criterion for correctness.

3.1 Closed information extraction

Task description. The goal of closed information triplet extraction (IE) is to extract a comprehensive set of facts from natural-language text. Formally, given a knowledge base represented by a knowledge graph (KG) containing a catalog of entities $\mathcal{E}$ and a catalog of relations $\mathcal{R}$, the goal is to extract the complete set $y_{\text{set}} \subset \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ of fact triplets expressed in a given input text x. It is crucial that the entities and relations in these triplets be accurately grounded in the KG's catalog. An example of this process can be seen in Fig. 1. The instructions I and I_{cg} for the sketcher and constrained decoder, respectively, are listed in Appendix D.1.

Constraints. We apply the constraints in Appendix D.2, which restrict entities (1.5 million) and relations (857) to the Wikidata KG, and enforce the structural constraint that outputs must be formatted as sequences of entity–relation–entity triplets.

Datasets and evaluation metrics. We use the Wiki-NRE (Trisedya et al., 2019) and SynthIE-text (Josifoski et al., 2023) datasets (details in Ap-

	Wiki-NRE			SynthIE-text		
	Precision	Recall	F1	Precision	Recall	F1
Without logit access						
GPT-4	42.4± 2.7	44.9±3.0	43.6± 2.6	46.1± 2.3	44.4± 2.2	45.2± 2.2
+ SketchGCD 7B	**38.7**± 3.2 (↓3.7)	**47.1**± 2.9 (↑2.2)	**46.1**± 2.8 (↑2.5)	**58.8**± 7.9 (↑12.7)	**47.3**± 2.3 (↑2.9)	**52.4**± 2.2 (↑7.2)
GPT-3.5-Turbo	27.4± 2.0	27.4± 2.5	27.4± 2.5	24.6± 2.0	23.1± 1.9	23.8± 1.9
+ SketchGCD 7B	**31.3**± 3.3 (↑3.9)	**46.4**± 2.8 (↑18.7)	**37.4**± 2.8 (↑10.0)	**49.5**± 2.8 (↑24.9)	**41.4**± 2.1 (↑18.3)	**45.1**± 2.1 (↑21.3)
Claude	34.1± 3.1	28.2± 2.8	30.8± 2.7	27.0± 2.0	26.7± 2.0	26.8± 2.0
+ SketchGCD 7B	**30.4**± 2.5 (↓3.7)	**40.6**± 2.8 (↑12.4)	**34.8**± 2.9 (↑4.0)	**51.4**± 2.5 (↑24.4)	**36.3**± 2.2 (↑9.6)	**42.5**± 2.2 (↑15.7)
Claude-instant	24.5± 2.9	18.0± 2.2	20.8± 2.4	13.0± 1.7	15.2± 1.6	14.0± 1.6
+ SketchGCD 7B	**44.9**± 3.3 (↑20.4)	**31.1**± 2.7 (↑13.1)	**36.7**± 2.5 (↑15.9)	**44.9**± 2.6 (↑31.9)	**31.1**± 2.1 (↑15.9)	**36.7**± 2.1 (↑22.7)
With logit access						
LLaMA-2-7B	18.3± 2.4	14.0± 1.8	15.9± 1.2	12.0± 1.5	8.6± 1.1	10.0± 1.3
+ SketchGCD 7B	**23.6**± 2.7 (↑5.3)	**34.2**± 2.9 (↑20.2)	**28.0**± 2.4 (↑12.1)	**33.3**± 2.5 (↑21.3)	**21.0**± 2.0 (↑12.4)	**25.7**± 2.1 (↑15.7)
+ CD	33.6± 2.7	32.9± 2.9	32.8± 2.5	34.0± 2.3	25.9± 2.0	29.4± 2.0
LLaMA-2-13B	22.6± 2.3	23.6± 2.4	23.1± 2.3	15.7± 1.6	12.7± 1.2	14.0± 1.5
+ SketchGCD 7B	**28.8**± 2.6 (↑6.2)	**44.2**± 3.0 (↑20.6)	**34.9**± 2.5 (↑11.8)	**36.1**± 2.0 (↑20.4)	**25.1**± 1.8 (↑12.4)	**29.6**± 1.8 (↑15.6)
+ CD	35.5± 2.6	39.1± 3.0	37.2± 2.5	39.7± 2.0	32.5± 1.8	35.7± 1.8
LLaMA-2-70B	26.1± 2.7	24.5± 2.3	25.7± 2.4	32.6± 2.0	26.9± 1.8	29.4± 1.8
+ SketchGCD 7B	**26.9**± 2.7 (↑0.8)	**41.0**± 2.6 (↑16.5)	**32.5**± 2.1 (↑6.8)	**52.0**± 2.0 (↑19.4)	**37.6**± 1.8 (↑10.7)	**43.6**± 2.0 (↑14.2)
+ CD	39.9± 2.6	46.5± 2.6	42.3± 2.1	62.7± 2.0	50.3± 2.0	55.8± 2.0

Table 1: **Results for closed information extraction,** in terms of triplet-based precision, recall, and F1-score (micro-averaged, with bootstrapped 95% confidence intervals) on the Wiki-NRE and SynthIE-text datasets. The results compare the effectiveness of SketchGCD (blue rows) against two baselines: (1) few-shot-prompted unconstrained decoding with powerful blackbox LLMs ("without logit access", white rows, Eq. 1) and (2) few-shot-prompted constrained decoding ("CD") with open-source LLMs ("with logit access", Eq. 3). Four demonstrations are used in few-shot prompting. LLaMA-7B serves as the constrained generator P_{cg} for SketchGCD.

pendix D.3). Performance is measured using micro precision, recall, and F1-score.

Results. We make the following observations based on Table 1: (1) The best blackbox LLMs (e.g., GPT-4) demonstrate strong performance even without constrained decoding, outperforming small open-source LLMs (LLaMA-2 7B/13B/33B) with constrained decoding. (2) Even without requiring access to logits, SketchGCD still manages to enhance the performance of LLMs significantly across all models of any size. (3) In case where logit access is available, constrained decoding is more effective than SketchGCD, as shown by the second half of the table. Given these observations, we conjecture that, if logits were accessible for blackbox LLMs, a further improvement in performance could be achieved with constrained decoding. However, without logit access, SketchGCD provides an effective alternative.

Impact of constrained decoder. We investigate the impact of the constrained decoder on the performance of SketchGCD. As shown in Table 2, given GPT-4 as the sketcher, the choice of the constrained decoder can affect the performance of SketchGCD. Contrary to our expectations, larger constrained decoder models do not always lead to better performance. Our intuition is that step 2 of SketchGCD (constrained decoding) is relatively simple, and the additional capacity of larger constrained decoders does not necessarily provide an advantage.

	Wiki-NRE			SynthIE-text		
	Prec	Recall	F1	Prec	Recall	F1
GPT-4	42.4	44.9	43.6	46.1	44.4	45.2
+ LLaMA-2-7B	38.7	**57.1**	46.1	**58.9**	47.3	52.4
+ LLaMA-2-13B	**42.9**	52.8	**47.3**	53.6	51.4	52.5
+ LLaMA-2-70B	35.2	54.0	42.6	58.1	**53.1**	**55.5**

Table 2: **Impact of constrained decoder model** (used in step 2 of SketchGCD) on closed information extraction. GPT-4 is used as the sketcher in all cases.

Impact of beam size. Our experiments show that using beam search is critical for the performance of both SketchGCD and classical constrained decoding. As shown in Table 3, employing beam search (even with a minimal beam size of 2) significantly improves performance over greedy decoding. Larger beam sizes further enhance performance, allowing the model to explore a larger search space, but with a diminishing returns.

The following example illustrates the importance of beam search. Suppose we are doing closed in-

	Wiki-NRE					
	LLaMA-2-7B + CD			LLaMA-2-13B + CD		
	Prec	Recall	F1	Prec	Recall	F1
1 beam	29.9	22.6	25.8	32.7	32.3	32.5
2 beams	33.6	32.1	32.8	35.9	**39.6**	37.7
4 beams	33.7	**32.9**	33.3	36.0	38.5	37.2
8 beams	**36.6**	30.8	**33.4**	**39.6**	36.0	**37.7**

Table 3: **Impact of beam size** in beam search on closed information extraction during classical constrained decoding. "1 beam" is equivalent to greedy decoding.

formation extraction on the sentence *"Mona Lisa is housed in the Musée du Louvre in Paris."* Our entity catalog contains among other, the entities *Louvre Museum* and *Musée d'Orsay*. During unconstrained decoding, the model might generate the following output with highest probability: *"[s] Mona Lisa [r] located in [o] Musée **du Louvre**"*. This output is invalid as the entity *Musée du Louvre* is not in the entity catalog and should be rendered as *Louvre Museum* instead.

With constrained decoding, the non-bold part of the output remains unaltered, as it satisfies the constraints. However, the bold suffix *"du Louvre"* is rejected by constrained decoding because *Musée du Louvre* is not in the entity catalog. The model will be forced to sample from the allowed entity catalog only, which can lead to *"Musée d'Orsay"* as the output. In this example, greedy constrained decoding was able to produce a valid yet incorrect output. On the contrary, had we used beam search, the model would have been able to consider both *Musée du Louvre* and *Louvre Museum* simultaneously, and would have been able to select the correct entity, *Louvre Museum*, for the output.

3.2 Constituency parsing

Task description. Constituency parsing involves breaking down a sentence into its syntactic components to form a parse tree that represents the sentence's structure. For instance, the sentence *"I saw a fox"* corresponds to the parse tree [S [NP [PRP *I*]] [VP [VBD *saw*] [NP [DT *a*] [NN *fox*]]]]. For a visual representation of this tree, see Appendix E Fig. 5. The instructions I and I_{cg} are listed in Appendix E.1.

Constraints. We apply the context-free grammar constraints in Appendix E.2 to ensure that brackets are balanced, and labels are consistent.

Dataset and evaluation metrics. Our evaluation uses the Penn Treebank test split. The parsing error rate of LLMs, regardless of size, is generally high, so we use only the shortest 25% of the samples for evaluation (up to 128 tokens according to the LLaMA tokenizer). We assess performance using bracketing recall and precision, as well as tag accuracy, as measured by the EVALB tool (Sekine and Collins, 2008). Since these metrics are only applicable to valid parse trees, and since models typically generate valid trees only for simpler inputs, one needs to be careful while interpreting the results, as weaker model may have better scores because they only generate a small fraction of valid parse trees (simpler ones) (Deutsch et al., 2019).

Results. The results in Table 4 show that even advanced LLMs like GPT-4 struggle to generate valid parse trees, especially for longer sentences. The following observations can be made: (1) Both SketchGCD and classical constrained decoding significantly help the model generate more structurally valid parse trees. (2) The other metrics mostly remain unchanged or slightly drop, as a larger validity rate means more difficult examples are included in the evaluation. (3) The most common errors in the unconstrained setting are *imbalanced brackets*, *invalid tags*, and *missing words,* as shown in Table 5. (4) With SketchGCD, the error rate for *imbalanced brackets* and *invalid tags* is significantly reduced, while the error rate for *missing words* increases significantly.

Note that constrained decoding with a more sophisticated grammar, as described in Appendix E.2, can achieve 100% valid trees and 100% valid tags (see Table 9). However, as implementing such a grammar is non-trivial, we use a simpler context-free grammar here (see Appendix E.2) to mimic the real-world scenario where a simpler might be preferred over a perfect grammar.

4 Related work

Constrained decoding. Deutsch et al. (2019) introduced a general constrained decoding framework for text generation based on automata. Scholak et al. (2021); Poesia et al. (2022); Geng et al. (2023) implemented incremental parsing for domain-specific tasks such as SQL generation. Beurer-Kellner et al. (2023); Poesia et al. (2023) have proposed iterative approaches to constrained decoding using blackbox LLM APIs, albeit with potential limitations such as excessive API calls (thus increasing monetary cost), as detailed in Appendix C.

Method	Bracket prec*	Bracket recall*	Bracket F1*	Tag accuracy*	Tag validity	Tree validity
Without logit access						
GPT-4	76.6± 5.0	67.7± 4.5	71.9± 4.0	95.4± 0.9	93.6± 4.0	86.0± 4.0
+ SketchGCD	**75.8± 2.4 (↓0.8)**	**67.8± 2.4 (↓0.1)**	**71.5± 2.4 (↓0.4)**	**95.3± 0.8 (↓0.1)**	**100± 0.0 (↑6.4)**	**92.5± 4.0 (↑6.5)**
GPT-3.5-Turbo	68.2± 0.7	55.5± 1.1	61.2± 0.6	93.1± 0.5	91.7± 2.4	76.9± 5.2
+ SketchGCD	**68.7± 3.2 (↑0.5)**	**56.6± 2.8 (↑1.1)**	**62.1± 2.0 (↑0.9)**	**92.6± 1.3 (↓0.5)**	**100± 0.0 (↑8.3)**	**81.5± 4.0 (↑4.6)**
Claude 2.1	73.1± 3.3	63.1± 2.6	67.7± 2.5	94.5± 1.1	95.1± 2.5	62.6± 5.2
+ SketchGCD	**71.6± 3.0 (↓1.5)**	**62.9± 2.5 (↓0.2)**	**66.9± 2.6 (↓0.8)**	**93.4± 1.3 (↓1.1)**	**100± 0.0 (↑4.9)**	**68.7± 5.5 (↑6.1)**
Claude-instant 1.2	71.3± 2.4	59.1± 1.4	64.7± 1.9	89.6± 1.6	91.7± 2.3	56.6± 5.2
+ SketchGCD	**66.6± 3.3 (↓4.7)**	**57.4± 3.1 (↓1.7)**	**61.6± 3.3 (↓3.1)**	**87.9± 2.5 (↓1.7)**	**100± 0.0 (↑8.3)**	**67.8± 3.7 (↑11.2)**
With logit access						
llama-2-7B	23.1± 4	10.4± 3	14.3± 4	14.9± 3	93.2± 3	32.1± 5
+ CD	28.5± 6 (↑5.4)	16.5± 3 (↑6.1)	20.9± 5 (↑6.6)	13.8± 2 (↓1.1)	100± 0 (↑6.8)	35.1± 5 (↑3.0)
llama-2-13B	33.4± 7	22.4± 4	26.8± 5	29.3± 4	95.5± 2	38.5± 6
+ CD	33.3± 6 (↓0.1)	21.8± 5 (↓0.6)	26.3± 5 (↓0.5)	34.0± 4 (↑4.7)	100± 0 (↑4.5)	43.4± 5 (↑4.9)
llama-2-70B	45.5± 6	37.7± 5	41.2± 5	55.5± 5	75.8± 5	40.4± 6
+ CD	39.8± 6 (↓5.7)	35.6± 4 (↓2.1)	37.6± 4 (↓3.6)	53.8± 4 (↓1.7)	100± 0 ((↑24.2))	47.6± 5 (↑7.2)

Table 4: **Results for constituency parsing,** in terms of bracketing precision, recall, F1-score, tag accuracy, tag validity, and parse tree validity (with bootstrapped 95% confidence intervals), on Penn Treebank test split. Only subset of samples whose ground-truth parse trees are shorter than 128 tokens (per LLaMA tokenizer) are considered (shortest 25% of the full dataset). Disclaimer: a weak method can have high precision by predicting very few valid parse trees (simple ones), and a strong method can have low precision by predicting more valid parse trees including complex ones (Deutsch et al., 2019). Four demonstrations are used in few-shot prompting. LLaMA-7B serves as the constrained generator P_{cg} for SGCD. (* Considering only sentences with valid parse trees.)

	Error type			
Method	InvalidTag	Extra	Imbal	Missing
GPT-4	6.4%	0.4%	10.2%	2.3%
+ SketchGCD	0.0%	0.0%	2.6%	6.0%
GPT-3.5-Turbo	8.3%	2.6%	9.4%	2.3%
+ SketchGCD	0.0%	1.5%	1.9%	16.2%
Claude 2.1	4.9%	3.8%	3.4%	30.2%
+ SketchGCD	0.0%	3.0%	3.8%	29.8%

Table 5: **Error analysis for constituency parsing** on the Penn Treebank dataset. *InvalidTag* refers to model generating invalid tags, *Extra* to model adding extra words absent from input, *Imbal* to model generating imbalanced brackets, and *Missing* to model dropping words from input.

Collaborative generation. Vernikos et al. (2023) and Welleck et al. (2023) explored training smaller language models to refine the outputs from larger models for enhanced quality. The skeleton-of-thought method (Ning et al., 2023) generates an initial output skeleton and then concurrently develops each segment. Grammar prompting (Wang et al., 2023) creates a meta-grammar to guide the output of LLMs in producing valid results.

5 Conclusion

So far, constrained decoding has been limited to open-source models that provide access to their logits during generation. Overcoming this limitation, we propose sketch-guided constrained decoding (SketchGCD), a simple method for constrained decoding with blackbox LLMs that does not require access to next-token logits during generation. By using separate sketching and refinement phases, SketchGCD allows to benefit from the power of blackbox LLMs while still enforcing constraints. Our work is complementary to existing methods for constrained decoding and can be used in conjunction with them. Despite its simplicity, SketchGCD achieves strong performance on tasks exhibiting strong structural constraints, outperforming unconstrained generation by a large margin.

6 Limitations

The limitations of our method include the following. First, SketchGCD adds an overhead as it requires a constrained decoder to refine the sketches after the sketching phase. Second, as LLMs keep getting better, the benefits of SketchGCD might diminish on some tasks as the unconstrained model's performance improves. Third, just as classical constrained decoding, SketchGCD can only enforce constraints at the structure level or the syntactic

level, but not at the semantic level. The model can still generate semantically incorrect outputs. However, in many real-world applications, we have observed semantic errors to be less common than structural errors.

Acknowledgments

We thank Grant Slatton for insightful discussions during the ideation phase. West's lab is partly supported by grants from Swiss National Science Foundation (200021_185043, TMSGI2_-211379), Swiss Data Science Center (P22_08), H2020 (952215), Microsoft Swiss Joint Research Center, and Google, and by generous gifts from Meta, Google, and Microsoft.

References

Lakshya A. Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K. Lahiri, and Sriram K. Rajamani. 2023. Guiding Language Models of Code with Global Context using Monitors.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting is programming: A query language for large language models. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1946–1969.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4.

Xiang Chen and Xiaojun Wan. 2023. A comprehensive evaluation of constrained text generation for large language models.

Sehyun Choi, Tianqing Fang, Zhaowei Wang, and Yangqiu Song. 2023. KCTS: Knowledge-Constrained Tree Search Decoding with Token-Level Hallucination Detection. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14035–14053, Singapore. Association for Computational Linguistics.

Daniel Deutsch, Shyam Upadhyay, and Dan Roth. 2019. A general-purpose algorithm for constrained sequential inference. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 482–492, Hong Kong, China. Association for Computational Linguistics.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10932–10952, Singapore. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

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

Wenyue Hua, Shuyuan Xu, Yingqiang Ge, and Yongfeng Zhang. 2023. How to index item ids for recommendation foundation models. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, SIGIR-AP '23. ACM.

Wenlong Huang, Fei Xia, Dhruv Shah, Danny Driess, Andy Zeng, Yao Lu, Pete Florence, Igor Mordatch, Sergey Levine, Karol Hausman, and Brian Ichter. 2023. Grounded Decoding: Guiding Text Generation with Grounded Models for Embodied Agents.

Pere-Lluís Huguet Cabot and Roberto Navigli. 2021. REBEL: Relation extraction by end-to-end language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2370–2381, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Martin Josifoski, Nicola De Cao, Maxime Peyrard, Fabio Petroni, and Robert West. 2022. GenIE: Generative information extraction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4626–4643, Seattle, United States. Association for Computational Linguistics.

Martin Josifoski, Marija Sakota, Maxime Peyrard, and Robert West. 2023. Exploiting asymmetry for synthetic training data generation: SynthIE and the case of information extraction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1555–1574, Singapore. Association for Computational Linguistics.

Tony Lee, Michihiro Yasunaga, Chenlin Meng, Yifan Mai, Joon Sung Park, Agrim Gupta, Yunzhi Zhang, Deepak Narayanan, Hannah Benita Teufel, Marco Bellagente, Minguk Kang, Taesung Park, Jure Leskovec, Jun-Yan Zhu, Li Fei-Fei, Jiajun Wu, Stefano Ermon, and Percy Liang. 2023. Holistic evaluation of text-to-image models.

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.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. 2023. Skeleton-of-Thought: Large Language Models Can Do Parallel Decoding. ArXiv:2307.15337 [cs].

Gabriel Poesia, Kanishk Gandhi, Eric Zelikman, and Noah D. Goodman. 2023. Certified deductive reasoning with language models.

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 *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Aarne Ranta. 2019. Grammatical framework: an interlingual grammar formalism. In *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*, pages 1–2, Dresden, Germany. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Satoshi Sekine and Michael Collins. 2008. Evalb: Bracket scoring program.

Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. 2022. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, 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 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7699–7715, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Bayu Distiawan Trisedya, Gerhard Weikum, Jianzhong Qi, and Rui Zhang. 2019. Neural Relation Extraction for Knowledge Base Enrichment. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 229–240, Florence, Italy. Association for Computational Linguistics.

Giorgos Vernikos, Arthur Bražinskas, Jakub Adamek, Jonathan Mallinson, Aliaksei Severyn, and Eric Malmi. 2023. Small language models improve giants by rewriting their outputs.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. Grammar prompting for domain-specific language generation with large language models.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Kexun Zhang, Hongqiao Chen, Lei Li, and William Wang. 2023. Syntax error-free and generalizable tool use for llms via finite-state decoding.

A Blackbox LLM logit access

Model	Logit bias	Token probs	MMLU
GPT-4-0614	Yes	Top 5	86.4
GPT-3.5-Turbo-0614	Yes	Top 5	70.0
Claude-2.1	No	No	78.5
Claude-instant	No	No	73.4
PaLM-2-text-bison	Yes	Top 5	78.3

Table 6: The blackbox LLMs we use in our experiments and the access they provide to the logit distribution. MMLU is the mainstream metric for LLM benchmarking.

Logit bias indicates whether the model's API allows user to pass in a logit bias vector to steer the decoding process, i.e., *write access* to the logit distribution. *Token probs* indicates whether the model's API allows user to access the model's next token probability distribution, i.e., *read access* to the logit distribution. MMLU (Hendrycks et al., 2021) is the mainstream metric for LLM benchmarking.

B Grammar constrained decoding

Grammar-constrained decoding takes a formal grammar G as input and ensures that the output string w is a valid *sentence* in the formal language $L(G)$ defined by the grammar G. This process is achieved through the integration of two key components: a *grammar completion engine* (Poesia et al., 2022) and a *sampling method*, e.g. greedy search, nucleus sampling, etc. The grammar completion engine is used to ensure the *grammaticality* of the output string, while the LLM is used to ensure the *plausibility* of the output string.

We use *Grammatical Framework*'s runtime powered completion engine (Ranta, 2019) with constrained beam search as the sampling method.

C Logit bias-based iterative decoding

Most blackbox LLM APIs do not provide complete access to the model's next token probability distribution at each decoding step. Nonetheless, many allow users to input a **logit bias** parameter to influence the decoding process, i.e., granting users *write access* but not *read access* to the model's logits at each decoding step. This parameter accepts a vector of logits that is added to the logits of the next token probability distribution at each decoding step. By using the logit bias parameter, users can direct the decoding process, effectively masking the logits of invalid tokens. This approach is particularly effective for static constraints, such as lexical constraints (Hokamp and Liu, 2017), where the constraints remain constant throughout the decoding.

However, the logit bias parameter is a static array and does not change during the decoding process. This makes it challenging to apply dynamic constraints, which change as decoding progresses, such as constraints involving membership in formal languages (Deutsch et al., 2019; Poesia et al., 2022; Geng et al., 2023).

A straightforward but costly solution for dynamic constraints is to iteratively invoke the blackbox LLMs API with updated logit bias vectors at each decoding step (Beurer-Kellner et al., 2023; Poesia et al., 2023; Agrawal et al., 2023; Choi et al., 2023). However, this approach is **prohibitively expensive**. Each API call generates only a single token, and the cost is calculated based on both the input and output tokens[2]. The expense of iteratively calling the blackbox LLMs API with new context and prefix at each step scales quadratically, being $O(n^2)$ where n is the length of the output sequence. Although methods like those proposed by Beurer-Kellner et al. (2023) and Poesia et al. (2023) use speculation to reduce the number of API calls, the costs can remain high, especially when the constraints are complex.

D Task 1. closed information extraction

In this section, we provide more details about the closed information extraction task.

D.1 Task instruction

We provide the instruction for the IE task in Figure 2. The few-shot demonstrations are rather long and thus we do not include them here. The full prompt is available in our code repository.

D.2 Grammar

The grammar is defined as follows, where V represents the set of variables, Σ the set of terminal symbols, and P the set of production rules:

$$V = \{S, T, A, B, C, E, R\}, \Sigma = \{\texttt{tokens}\}$$
$$P = \{S \rightarrow [ST|\epsilon], T \rightarrow [ABC\,\texttt{[e]}]$$
$$A \rightarrow [\texttt{[s]}\ E], E \rightarrow (\texttt{entity1}|\texttt{entity2}|...),$$
$$B \rightarrow [\texttt{[r]}\ R], R \rightarrow (\texttt{rel1}|\texttt{rel2}|...)$$
$$C \rightarrow [\texttt{[o]}\ E], \epsilon \rightarrow \texttt{</s>}\}$$

[2] See https://openai.com/pricing for details.

Extract the subject-relation-object triples in fully-expanded format from texts below. The subjects and objects are entities in Wikidata, and the relations are Wikidata properties. Here are a few examples.

(a) Instruction for sketcher

In this task, you will be provided with texts along with draft annotations that represent extracted information triples in the form of subject-relation-object. Your role is to refine these triples to ensure completeness and accuracy. Here are a few examples.

(b) Instruction for constrained decoder

Figure 2: Instructions for parsing tasks.

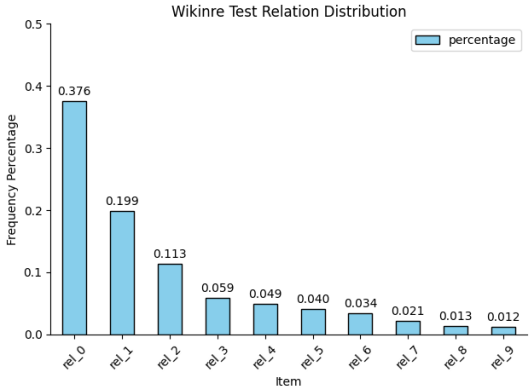
The outputs are structured as a sequence of triplets, where each triplet is separated by a special marker `[e]`. Every triplet consists of a subject, a relation, and an object. These elements are each preceded by a special marker: `[s]` for the subject, `[r]` for the relation, and `[o]` for the object, respectively. The subject and object are pre-defined Wikidata entities, and the relation is a pre-defined Wikidata property. This grammar is classified as context-free, more specifically, as a regular grammar.

D.3 IE datasets

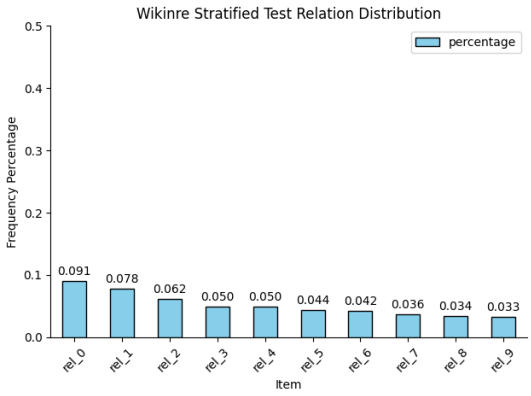
The original SynthIE-text and Wiki-NRE datasets comprise 50,000 and 30,000 samples, respectively. To minimize the evaluation cost on Large Language Models (LLMs), we use a smaller subset consisting of 1,000 samples from each dataset.

As noted by Josifoski et al. (2023), the Wiki-NRE dataset displays a significant skew in its relations distribution: the top 10 relations constitute 92% of the triplets, with the top 3 alone accounting for 69%. To ensure our test set accurately reflects the overall dataset, we have downscaled it to 1,000 samples to balance the distribution of relations, as shown in Fig. 3

The SynthIE-text dataset, synthesized by reverse prompting **Text-Davinci-003** with triplets from Wikidata, stands out due to its substantial size, diverse content, and high-quality human ratings, as highlighted in (Josifoski et al., 2023). This contrasts with prior datasets such as REBEL (Huguet Cabot and Navigli, 2021), whose annotation quality is low (Josifoski et al., 2022). However, a potential minor bias may exist towards GPT-4 and GPT-3.5-Turbo, as SynthIE-text was generated from a model in their family, **Text-Davinci-003**. Despite this, we maintain that this does not compromise the validity of our method, given that our primary focus is on the comparative performance with and without the application of SketchGCD.



(a) Original relation distribution in WikiNRE test set



(b) Stratified relation distribution in WikiNRE test set

Figure 3: Relation distribution in WikiNRE before and after stratification.

D.4 Discussion of GPT-4 on cIE

An intriguing finding is that SketchGCD's performance on the SynthIE-text dataset using GPT-4 (F1=45.6) is marginally lower than that achieved with few-shot prompting alone, without constrained decoding (F1=45.8). Our analysis suggests that the constrained decoder occasionally struggles to adhere to the sketcher's outline, resulting in fewer triplets than expected in the output. This observation is consistent with Wang et al. (2023)'s findings, where constrained decoding was noted to reduce the diversity of the generated samples.

More critically, since SynthIE-text is synthetically generated by reverse prompting **Text-Davinci-**

	Ratio of invalid triplets					
	Wiki-NRE			SynthIE-text		
	Entity	Rel	Triplet	Entity	Rel	Triplet
GPT-4	**19.4**	**21.9**	**45.2**	**7.6**	**28.1**	**37.3**
GPT-3.5-Turbo	23.4	50.2	65.8	13.8	52.5	63.3
Claude	17.0	41.1	55.5	17.4	52.8	64.6
Claude-ins	19.6	48.4	62.6	13.8	43.3	52.7
SketchGCD	0	0	0	0	0	0

Table 7: **Triplets grounding analysis.** We report the percentage of generated entities, relations, and triplets that are not present in the knowledge catalogue in few-shot unconstrained setting. The grounding precision for **constrained** methods is 100% by construction, and thus 0% invalid triplets.

003 with triplets from Wikidata, its text doesn't exhibit the naturalness characteristic of the Wiki-NRE dataset. For instance, sentences in SynthIE-text often resemble direct copies with slight alterations from the original entity and relation names. This tendency facilitates the LLMs' task of grounding entities and relations in the Knowledge Graph (KG), thereby diminishing the necessity for constrained decoding.

However, in real-world scenarios, text is typically more intricate, and grounding entities and relations in the KG is not as straightforward. Despite this, the overall performance enhancement provided by SketchGCD across various models remains noteworthy, averaging gains of up to 10.7% and 8.1% on Wiki-NRE and SynthIE-text, respectively.

D.5 Grounding analysis

In this study, we delve into the grounding efficacy of GPT-4's output. A triplet is deemed grounded when both its subject and object entities, as well as the relation, are present in the KG. Furthermore, for a grounded triplet to be considered correct, it must also be part of the target triplet set.

Given that being grounded is essential but not solely adequate for being correct, it is crucial to assess how well GPT-4's output aligns with the KG. According to the data presented in Table 7, we observe that a significant portion of the output triplets from GPT-4 are not grounded in the KG, amounting to 45% and 37% on the Wiki-NRE and SynthIE-text datasets, respectively. This finding sheds light on the importance of constrained decoding, as it ensures that the output is grounded in the KG, thereby increasing the likelihood of validity.

E Task 2. constituency parsing

In this section, we provide more details about the constituency parsing task.

E.1 Task instruction

We provide the instruction for the CP task in Figure 4. The few-shot demonstrations are rather long and thus we do not include them here. The full prompt is available in our code repository.

Perform constituency parsing on the provided sentences in accordance with the Penn TreeBank annotation guidelines. Here are a few examples.

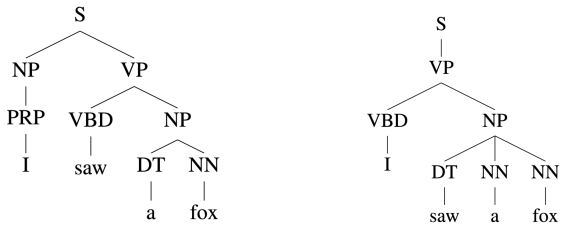
(a) Instruction for sketcher

In this task, you will be provided with a draft annotations that represent the parse tree of a sentence in Penn TreeBank format. Your task is to rewrite the parse tree and fix error if any. Here are a few examples.

(b) Instruction for constrained decoder

Figure 4: Instructions for parsing tasks.

E.2 Constraints and grammar



(a) The correct constituency parse tree

(b) A grammatical but incorrect parse tree

Figure 5: Parse trees for the sentence "I saw a fox".

Here we describe the grammar used to constrain the generative constituency parsing task.

Linearization. A constituency parse tree is inherently a recursive structure. To effectively represent this tree as a sequence of tokens for generation by a Large Language Model (LLM), a linearization is required. Two common strategies for this linearization are *pre-order traversal* and *post-order traversal*.

We have chosen to adopt the pre-order traversal strategy. This approach is also the default method

used in the PYEVALB tool (Sekine and Collins, 2008) and in the construction of the Penn Treebank (Marcus et al., 1993). As an illustration, the parse tree in Fig. 5a is linearized in the following format: [S [NP [PRP *I*]] [VP [VBD *saw*] [NP [DT *a*] [NN *fox*]]]].

The linearised parse tree needs to satisfy the following structural constraints:

- *Completeness*: Every word in the sentence needs to be included in the parse tree.
- *Balanced brackets*: At any point in the linearized parse tree, the right bracket `]` should close a previously unclosed left bracket `[` and every left bracket `[` should be eventually closed by a right bracket `]`.
- *Label consistency*: The label of terminal and non-terminal nodes needs to be consistent with the Penn Treebank format.

Simple Context-Free Grammar. The tree structure of the parse tree is usually captured by a context-free grammar as shown in Table 8.

```
root ::= tree;
tree ::= node;
node ::= clause | phrase | word;
clause ::= spaced_open_parenthesis, space,
            clause_tag, function_tag*,
            index?, node*,
           spaced_close_parenthesis;
phrase ::= spaced_open_parenthesis, space,
            phrase_tag, function_tag*,
            index?, node*,
            spaced_close_parenthesis;
word ::= spaced_open_parenthesis, space,
        word_tag, space, actual_word,
        spaced_close_parenthesis;

clause_tag ::= "S" | ...  | "SQ";
phrase_tag ::= "ADJP" | ...| "WHADVP";
word_tag ::= "CC" |...|"WRB";

function_tag ::= "-ADV" |... | "-TTL";
actual_word ::= "xxx";
index ::= "-", [1-9], {0-9};
spaced_open_parenthesis ::= space, "(";
spaced_close_parenthesis ::= space, ")";
space ::= " ";
```

Table 8: Lite Context-Free Grammar for constituency parsing.

Sophisticated Regular Grammar. However, the context-free grammar is not sufficient to capture the *completeness* constraint, motivating the use of a more restrictive grammar. Geng et al. (2023) proposed a sophisticated regular grammar to enforce

$$
\begin{aligned}
&S \to B_{0,0} \\
&B_{i,j} \to [\alpha\,(B_{i,j+1} \mid C_{i,j+1})]; \\
&C_{i,j} \to x_i\,(C_{i+1,j} \mid E_{i+1,j}); \\
&C_{n,j} \to E_{n,j}; \\
&E_{i,j+1} \to]\,(E_{i,j} \mid B_{i,j}); \\
&E_{n,j+1} \to]\,E_{n,j}; \\
&E_{n,0} \to \varepsilon; \\
&\text{where}\,\alpha = (S \mid NP \mid VP \mid \ldots) \quad \text{and}\, x_i \in \text{tokens}
\end{aligned}
$$

Figure 6: Sophisticated Regular Grammar for constituency parsing.

the constraints of *completeness*, *balanced brackets*, and *label consistency* as shown in Fig. 6.

The grammar falls into the category of *regular grammar* and is *input-dependent*. it reproduces the input sentence, represented as a sequence $x = \langle x_0, \ldots, x_{n-1} \rangle$ of words, in left-to-right order, interspersing it with node labels and balanced brackets. In order to guarantee balanced brackets, the non-terminals $B_{i,j}$ count the number of opened left brackets `[` using the second subscript index j, and the rules ensure that the number of closed brackets can never exceed the number of previously opened brackets.

F Data contamination risk

There is a rising concern over the data contamination risk of evaluating LLMs on downstream tasks. The datasets of our experiments are publicly available on internet so there is a risk that the models may have seen the data, such as the ground true parse tree of Penn Treebank during pretraining. However, the risk of data contamination is independent of our method and doesn't affect the validity of our conclusions.

Method	Bracket-Prec	Bracket-Recall	Bracket-F1	Tag Accuracy	Valid Tag	Valid Tree
GPT-4	76.6	67.7	71.9	95.4	93.6	86.0
+ **Lite Context-Free Grammar**	**75.8**	**67.8**	**71.5**	**95.3**	**100**	**92.5**
+ **Sophisticated Regular Grammar**	**69.3**	**63.1**	**66.1**	**98.5**	**100**	**100**
GPT-3.5-Turbo	68.2	55.5	61.2	93.1	91.7	76.9
+ **Lite Context-Free Grammar**	**68.7**	**56.6**	**62.1**	**92.6**	**100**	**81.5**
+ **Sophisticated Regular Grammar**	**61.2**	**49.4**	**54.7**	**96.0**	**100**	**100**
Claude	73.1	63.1	67.7	94.5	95.1	62.6
+ **Lite Context-Free Grammar**	**71.6**	**62.9**	**66.9**	**93.4**	**100**	**68.7**
+ **Sophisticated Regular Grammar**	**52.1**	**45.4**	**48.5**	**75.9**	**100**	**99.2**
Claude-instant	71.3	59.1	64.7	89.6	91.7	56.6
+ **Lite Context-Free Grammar**	**66.6**	**57.4**	**61.6**	**87.9**	**100**	**67.8**
+ **Sophisticated Regular Grammar**	**59.6**	**49.2**	**53.9**	**84.9**	**100**	**99.5**

Table 9: **Constituency parsing with two different grammar constraints**, measured in terms of bracketing recall, precision, F1-score, and tag accuracy (with bootstrapped 95% confidence intervals) †Only subset of samples whose ground-truth parse trees are shorter than 128 tokens(LLaMAtokenizer) are considered, which accounts for shortest 25% of the samples.