Learning and Enforcing Context-Sensitive Control for LLMs

Mohammad Albinhassan[1], Pranava Madhyastha[2,4], Mark Law[3], Alessandra Russo[1,4]

[1]Imperial College London, [2]City University of London, [3]ILASP Limited, UK
[4]The Alan Turing Institute
{m.albinhassan23, a.russo}@imperial.ac.uk,
pranava.madhyastha@city.ac.uk, mark@ilasp.com
Correspondence: m.albinhassan23@imperial.ac.uk

Abstract

Controlling the output of Large Language Models (LLMs) through context-sensitive constraints has emerged as a promising approach to overcome the limitations of Context-Free Grammars (CFGs) in guaranteeing generation validity. However, such constraints typically require manual specification—a significant barrier demanding specialized expertise. We introduce a framework that automatically learns context-sensitive constraints from LLM interactions through a two-phase process: syntactic exploration to gather diverse outputs for constraint learning, followed by constraint exploitation to enforce these learned rules during generation. Experiments demonstrate that our method enables even small LLMs (1B parameters) to learn and generate with perfect constraint adherence, outperforming larger counterparts and state-of-the-art reasoning models. This work represents the first integration of context-sensitive grammar learning with LLM generation, eliminating manual specification while maintaining generation validity.

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing, demonstrating unprecedented capabilities across diverse domains (Brown et al., 2020; Dubey et al., 2024). However, ensuring correctness in LLM outputs remains a critical challenge, particularly when outputs must adhere to specific formal constraints. While recent advances in controlled decoding have enabled enforcement of syntactic correctness through Context-Free Grammars (CFGs) (Geng et al., 2023; Beurer-Kellner et al., 2024; Park et al., 2024, interalia), ensuring semantic validity requires additional mechanisms.

The fundamental limitation lies in the expressivity gap between CFGs and real-world requirements. Many domains demand not only local structural correctness but also relationships between distant elements in a sequence, nested structures, and so on (Scholak et al., 2021). Such constraints can only be expressed by more powerful formalisms like Context-Sensitive Grammars (CSGs). For instance, a CFG may capture the language $a^i b^j c^k$, where any number of a's must be followed by any number of b's and then c's, but only a CSG can capture dependencies such as equal counts, i.e., $a^n b^n c^n$. Consequently, domain-specific solutions were proposed for tasks like semantic parsing (Lei et al., 2025; Poesia et al., 2022; Roy et al., 2023), and later, general domain-independent frameworks have been developed (Albinhassan et al., 2025) to broaden applicability. However, a barrier to adoption exists, as formal specifications for context-sensitive constraints demand expertise that may not be readily available. This contrasts with CFGs, which are more widely accessible for many structured generation tasks (Wang et al., 2023).

We introduce a framework that automatically learns context-sensitive constraints from LLM outputs. Our approach operates in two phases (Figure 1): (1) *syntactic exploration*, where we leverage a CFG-constrained temperature-sampling mechanism to collect diverse syntactically valid outputs, which are then labeled by an oracle and used to learn context-sensitive constraints through a logic-based learner; and (2) *constraint exploitation*, where these learned constraints control LLM generation to guarantee context-sensitive correctness. This represents the first integration of context-sensitive grammar learning with LLM generation.

Our empirical results on synthetic grammar synthesis tasks demonstrate our framework can successfully learn the ground-truth context-sensitive constraints via LLM interactions. As such, our approach induces control in LLM generations and guarantees constraint adherence for even small models (i.e., 1B parameters) — a capability even state-of-the-art reasoning models (i.e., DeepSeek-R1 (Guo et al., 2025)) fail to achieve consistently.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), pages 834–842
July 28-29, 2025

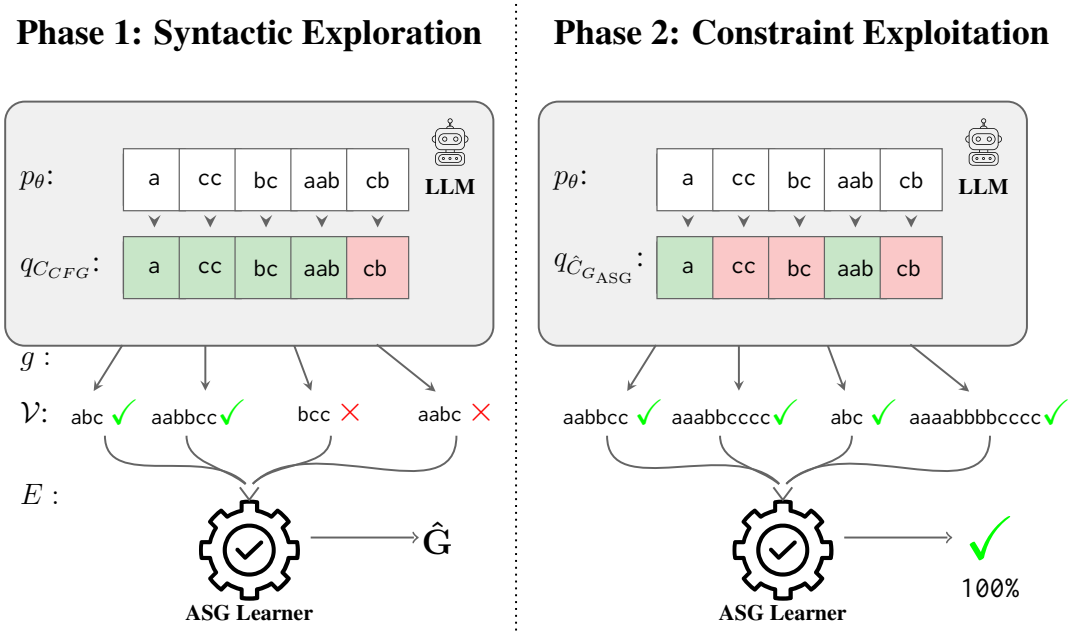

Figure 1: Two-phase methodology for learning context-sensitive constraints. **Phase 1:** An LLM p_θ samples diverse sequences using generator g from a CFG-masked distribution ($q_{C_{\text{CFG}}}$). Tokens such as bc are CFG-valid but context-sensitively invalid, leading to oracle rejection ($\mathcal{V}$) and red masking in $\hat{C}_{\text{ASG}}$. Valid tokens appear green, invalid tokens red. Labeled examples form dataset E for the ASG learner to construct $\hat{G}$. **Phase 2:** The LLM uses the learned ASG-constrained distribution ($q_{\hat{C}_{G_{\text{ASG}}}}$), disallowing tokens that may lead to violations (red), while valid tokens remain accessible (green), ensuring all outputs satisfy the target grammar (✓). The gear in Phase 2 illustrates all constraints have been learned (100%), so nothing new is learned (note: this is for visualization purposes only).

2 Related Work

Significant work in controlled decoding has focused on CFG-based approaches (Beurer-Kellner et al., 2023; Willard and Louf, 2023, interalia), where LLM generations must conform to the grammar's specification (Welleck et al., 2024). These methods address syntactic validity but are unable to enforce context-sensitive constraints critical for many real-world tasks. Semantic parsing via LLMs aim to capture such constraints; however, they employ domain-specific rules (Scholak et al., 2021; Roy et al., 2023; Poesia et al., 2022). Recent work develops a unifying domain-independent framework for controlling LLM outputs according to CSGs and semantic constraints via Answer Set Grammars (ASGs) (Albinhassan et al., 2025), though these constraints remain handcrafted.

Wang et al. (2023) propose grammar prompting, where an LLM predicts CFGs for specific tasks to control generation. However, the approximated grammar remains context-free and may be incorrect. In contrast, we extend Albinhassan et al. (2025) by automatically learning context-sensitive constraints expressed as formal annotations over CFGs. These constraints are learned via a state-of-the-art logic-based learner using LLM-generated examples labeled by an oracle. Thus, adapting to new tasks without handcrafting constraints with guaranteed correctness on the learned grammar.

3 Background

Formal Languages A formal language $L \subseteq \Sigma^*$ is a set of strings composed of a vocabulary Σ. L is generated by a grammar $G = \langle N, T, P, S \rangle$ where N are non-terminals, $T = \Sigma$ are terminals, P are production rules, and $S \in N$ is the start symbol. CFGs compose of rules of the form $A \to \alpha$ where $A \in N, \alpha \in (N \cup T)^*$, allowing them to capture syntax. While CSGs encode rules of the form $\alpha A \beta \to \alpha \gamma \beta$ where $A \in N$, $\alpha, \beta \in (N \cup T)^*$, $\gamma \in (N \cup T)^+$. Hence, CSGs can capture context-dependent patterns (Linz and Rodger, 2022). As such, while a CFG captures $L_1 = \{a^i b^j c^k : i, j, k \geq 0\}$, only a CSG can express $L_2 = \{a^n b^n c^n : n \geq 0\}$.

Answer Set Grammars ASGs (Law et al., 2019) extend production rules of CFGs with context-sensitive constraints expressed in a logic-based language called ASP (Lifschitz, 2019). A string w belongs to the language represented by an ASG

G_{ASG}, i.e., $w \in L(G_{\text{ASG}})$, if there exists a parse tree derivation whose logic representation (in ASP) is satisfiable — meaning a set of logical statements, rules, or constraints must all be true simultaneously. For instance, the CFG component of an ASG captures L_1, and the context-sensitive annotations capture L_2 by imposing constraints on the number of occurrences of terminal symbols. These annotations have been shown to be learnable from positive and negative examples of a CSG using the logic-based learner ILASP (Law et al., 2014). For example, given L_1, a positive example (i.e., `aabbcc`) and a negative example (i.e., `aabc`), ILASP learns constraints for equal counts of `a`'s, `b`'s, and `c`'s.

4 Methodology

Our approach learns context-sensitive constraints for language model generation through a two-phase process: *syntactic exploration* and *constraint exploitation*. Syntactic exploration works as follows: (1) Starting with a CFG, we generate diverse samples from a syntactically constrained LLM via temperature-sampling (we alter temperature to obtain diverse sequences (Renze, 2024)); (2) We use an oracle to label the samples into positive ($w \in L(G_{\text{CSG}})$) and negative ($w \notin L(G_{\text{CSG}})$) sets; (3) We feed the labeled examples to the ASG learner to learn the context-sensitive annotations over the given CFG that covers all samples. For constraint exploitation, we follow Albinhassan et al. (2025) to constrain the LLM's generation to conform to the learned context-sensitive constraints.

4.1 Syntactic Exploration

(1) CFG-Constrained Diverse Sampling. To learn the context-sensitive constraints of a target grammar G_{ASG}, we require samples that both satisfy and violate these constraints while maintaining syntactic validity (Figure 1, left). Let p_θ denote a language model with parameters θ that defines a distribution over tokens $p_\theta(y_t|x, y_{<t})$ given input x and context $y_{<t}$. We seek to learn the grammar $\hat{G}_{\text{ASG}}$ by collecting a dataset $\mathcal{D}$ containing both positive ($y \in L(G_{\text{ASG}})$) and negative examples ($y \in L(G_{\text{CFG}}) \setminus L(G_{\text{ASG}})$) of the underlying context-sensitive constraints.

Following Albinhassan et al. (2025), we define a constraint function $\mathcal{C} : \mathcal{V}^* \to 2^{\mathcal{V}}$ that maps any prefix $y_{<t} = (y_1, \ldots, y_{t-1}) \in \mathcal{V}^*$ to the set of valid next tokens according to a grammar G:

$$\mathcal{C}(y_{<t}) = \{y_t \in \mathcal{V} \mid \exists\, w \in L(G) : (y_{<t} \circ y_t) \text{ is a prefix of } w\} \quad (1)$$

where $\circ$ denotes token concatenation and $\mathcal{V}$ is the vocabulary of the language model's tokenizer.

We define a temperature-based syntactically constrained sampling generator to construct $\mathcal{D}$ with sufficient diversity to capture various context-sensitive violations. The sampling generator g with parameters $\phi = \{\mathcal{T}, N, C_{\text{CFG}}\}$ is:

$$g(y|x; p_\theta, \phi) = \{y^{(n,k)} \sim q_{\mathcal{C}_{\text{CFG}}}(\cdot \mid x; p_\theta, \tau_k), \; n \in [N], k \in [|\mathcal{T}|]\} \quad (2)$$

where each $y^{(n,k)}$ is a generated sequence, $\mathcal{C}_{\text{CFG}}$ the constraint function for grammar G_{CFG}, N is the number of sequences per temperature value, and $\mathcal{T} = \{\tau_1, \ldots, \tau_T\}$ is the temperature schedule.

Each sequence is sampled as $y \sim q_{\mathcal{C}_{\text{CFG}}}$, where:

$$q_{\mathcal{C}_{\text{CFG}}}(y_t \mid x, y_{<t}; p_\theta, \tau) \propto \exp\left(\frac{s_\theta(y_t \mid x, y_{<t})\, \mathbb{I}[y_t \in \mathcal{C}_{\text{CFG}}(y_{<t})]}{\tau}\right) \quad (3)$$

where s_θ is the model logit function, τ is the temperature parameter, and $\mathbb{I}(\cdot)$ is the indicator function. This guarantees that any sampled sequence belongs to $L(G_{\text{CFG}})$.

For a given task with M problem instances $\{x_i\}_{i=1}^{M}$, applying this generator to all $x_i \in M$ yields a dataset $\mathcal{D} = \{y_{i,j,k} : i \in [M], j \in [N], k \in [T]\}$, where $|\mathcal{D}| = M \cdot N \cdot |\mathcal{T}|$.

(2) Oracle Labeling. We employ a task-specific oracle $V : \Sigma^* \to \{0, 1\}$ to annotate each generated sequence. The oracle is treated as a deterministic ground truth labeler for the constraints, returning $V(y) = 1$ if y satisfies all constraints and 0 otherwise. This transforms our dataset into:

$$E = \{(y_{i,j,k}, V(y_{i,j,k})) : y_{i,j,k} \in \mathcal{D}\} \quad (4)$$

The diversity in temperature sampling ensures positive and negative examples are sufficiently populated, providing the ASG learner with comprehensive coverage of the constraint space.

(3) Constraint Learning via ASG Learner. We segment E into E^+ and E^-, containing samples conforming to and violating the constraints, respectively, as given by the oracle. We feed as input

to the ASG learner G_{CFG}, E^+, and E^-. Consequently, $\hat{G}_{\text{ASG}}$ is constructed by learning the ASP annotations over G_{CFG} such that $\hat{G}_{\text{ASG}}$ covers all samples in E (see Appendix A for formal details).

4.2 Constraint Exploitation

With the learned ASG $\hat{G}_{\text{ASG}}$, we transition from syntactic exploration to constraint exploitation (Figure 1, right). Following Albinhassan et al. (2025), we sample sequences $y \sim q_{\hat{C}_{\text{ASG}}}$ encoding the constraint function $\hat{C}_{\text{ASG}}$ for the learned grammar $\hat{G}_{\text{ASG}}$. This is similar to Equation (3) without temperature variations. At this point, the model has no further access to the oracle, relying entirely on the learned constraints to ensure context-sensitive validity.

5 Experiments

5.1 Task Definition

We evaluate our approach on two synthetic grammar synthesis tasks, where the LLM must generate strings from a target context-sensitive language. Following Albinhassan et al. (2025), we adopt $L_1 = \{a^n b^n c^n \mid n \geq 1\}$ and craft $L_2 = \{a^n b^n c^m \mid n, m \geq 1\}$. Each problem instance $x_i \in M$ prompts the LLM to generate strings with various values of n and m, producing diverse examples that capture both valid and invalid patterns with respect to the context-sensitive constraints for the ASG learner.

5.2 Experimental Setup

Models. We evaluate closed- and open-source models across various sizes: GPT-4.1, o1, o3-mini, o4-mini, and DeepSeek-R1 through their respective APIs, and Llama models (3.2 1B, 3.2 3B, 3.1 8B, and 3.1 70B) which we run locally (see Appendix C for GPU cluster details). All models are prompted identically using few-shot examples.

ASG Learning Configuration. We sample 10 generations at each temperature value $\tau \in \{0, 0.1, ..., 1.0\}$ for the syntactic exploration phase to construct a diverse dataset $\mathcal{D}$. The oracle $V(y)$ is implemented as a Python program to check constraint validity, i.e., checks the counts of `a`'s, `b`'s, and `c`'s and their respective ordering. The ASG learner constructs $\hat{G}_{\text{ASG}}$ by learning the ASP annotations over G_{CFG} from these examples segmented into E^+ and E^-.

Unconstrained and Constraint Exploitation Sampling Mechanisms. For API-based models, we use their standard generation settings. For Llama models, we employ three sampling approaches: (1) unconstrained rejection sampling, where we generate 50 samples and select a generation based on the oracle's feedback; and constrained generation, where we apply (2) the learned ASG and (3) a handcrafted ASG for comparison with Albinhassan et al. (2025).

Model	G	Accuracy	
		$a^n b^n c^n$	$a^n b^n c^m$
GPT 4.1	-	63.3%	76.7%
o1	-	86.7%	96.7%
o3 mini	-	63.3%	86.7%
o4 mini	-	90.0%	93.3%
DeepSeek-R1	-	80.0%	86.7%
Llama 1B	-	20.0%	6.7%
Llama 1B	$\mathbf{G_{ASG}}$	**100.0%**	**100.0%**
Llama 1B	$\mathbf{\hat{G}_{ASG}}$	**100.0%**	**100.0%**
Llama 70B	-	76.7%	53.3%
Llama 70B	$\mathbf{G_{ASG}}$	**100.0%**	**100.0%**
Llama 70B	$\mathbf{\hat{G}_{ASG}}$	**100.0%**	**100.0%**

Table 1: Accuracy results for $a^n b^n c^n$ and $a^n b^n c^m$ with different LLMs (Model) and grammar constraints (G).

Evaluation Metrics. We evaluate methods using context-sensitive validity accuracy, defined as the percentage of generated sequences that belong to the ground-truth grammar G_{ASG}.

5.3 Results and Analysis

Table 1 summarizes our findings across models and constraints (see Appendix B for results on 3B and 8B). We analyze two key aspects: the effectiveness of our ASG learning approach, and the impact of learned constraints on accuracy.

Ground-Truth ASGs are Learned. Table 1 showcases that constraining LLM p_θ with the ground-truth grammar (G_{ASG}) and the learned grammar ($\hat{G}_{\text{ASG}}$) both provide 100% accuracy and conform to all constraints. Whilst it could be the case that our sampling mechanism with the ASG learner only learned a subset of constraints sufficient for the LLM not to make any errors, i.e., the LLM already captures some of these via the prompt, manual inspection confirmed $\hat{G}_{\text{ASG}}$ is identical to G_{ASG}. The reasons behind this are twofold: (1) our syntax-constrained temperature-based sampling approach effectively covers the space of context-sensitive constraints sufficiently, i.e., the necessary positive and negative examples;

(2) the ASG learner based on ILASP guarantees that all examples will be covered, and if a solution exists, it will be found (see Law et al. (2015) for soundness and completeness proofs).

Guaranteed Correctness via Constraints. When applying the learned ASG constraints during generation, all models—even the smallest 1B-parameter model—achieve 100% accuracy on both context-sensitive tasks. In contrast, unconstrained generation with larger and closed-source models fails to provide such guarantees, with Llama 70B achieving only 76.7% and 53.3% accuracy, and GPT-4.1 obtaining 63.3% and 76.7% on L_1 and L_2, respectively. Although increasing the scale of model parameters improves performance (e.g., Llama 1B's 20.0% and 6.7% vs. Llama 70B), unconstrained models still lack reliability and robustness in generation.

Despite employing significantly more computational resources through extended reasoning steps (Valmeekam et al., 2025; Guo et al., 2025; Albinhassan et al., 2025), state-of-the-art reasoning models (i.e., o1, DeepSeek-R1, etc.) still produce invalid sequences. Consider o4-mini, the best performing unconstrained model, still only achieves 90.0% and 93.3% on L_1 and L_2, respectively. These results demonstrate that our neuro-symbolic constraint learning approach provides correctness guarantees that cannot currently be achieved through scale or inference time multi-step reasoning alone. Most notably, a 1B-parameter model eliminates the need for handcrafted constraints by learning and enforcing the ground-truth constraints, consistently outperforming all unconstrained models. This emphasizes the complementary strengths of neural language generation and symbolic constraint enforcement.

6 Conclusion and Future Work

We presented a novel framework for automating the learning of context-sensitive constraints for controlled LLM generation. The synergistic combination of syntactic exploration and constraint exploitation eliminates the need for manual constraint specification while maintaining correctness guarantees. Our empirical results demonstrate that this method enables small LLMs to learn and generate with perfect constraint adherence, outperforming larger and specialized reasoning models.

We plan to extend our work to real-world settings where constraints represent semantic relationships with intrinsic meaning (i.e., semantic parsing, agent planning). We further aim to explore active learning settings using ASG's sample-efficient one-shot learning ability. Thus, enabling continuous constraint refinement in lifelong learning tasks where a complete ASG may not be initially captured.

Limitations

Our approach demonstrates promising results, yet several limitations warrant consideration. First, the syntactic exploration phase lacks formal convergence guarantees. While temperature-based sampling empirically captured sufficient constraint violations in our synthetic domains, we cannot guarantee comprehensive coverage of larger constraint spaces. Establishing theoretical connections between sampling strategies, sample efficiency, and constraint space coverage remains an open challenge.

Second, our framework currently addresses only hard constraints where outputs are strictly valid or invalid. Many real-world NLP tasks, such as machine translation or question answering, involve soft constraints where outputs exist on a spectrum of acceptability. This binary classification approach limits applicability to domains requiring nuanced evaluation of correctness.

Third, our method assumes the underlying language model has been trained on data containing the relevant terminals and has developed statistical priors aligned with the target formal languages. For domains with limited representation in the training corpus, the generated samples may be insufficient to capture the full spectrum of context-sensitive constraints. We acknowledge these limitations and aim to address them in our future work (Section 6).

7 Acknowledgements

We thank Microsoft Research - Accelerating Foundation Models Research program for the provision of Azure resources to run some of the LLMs used in the experiments in this paper. This research was partially sponsored by DEVCOM Army Research Lab under W911NF2220243, EPRC project EP/Y037421/1, and The Alan Turing Institute's project on Robust Inference with PASP scaffolds for LLMs.

References

Mohammad Albinhassan, Pranava Madhyastha, and Alessandra Russo. 2025. Sem-ctrl: Semantically controlled decoding. *arXiv preprint arXiv:2503.01804*.

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.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2024. Guiding LLMs the right way: Fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 3658–3673. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. 2019. Representing and learning grammars in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2919–2928.

Mark Law, Alessandra Russo, and Krysia Broda. 2014. Inductive learning of answer set programs. In *Logics in Artificial Intelligence*, pages 311–325, Cham. Springer International Publishing.

Mark Law, Alessandra Russo, and Krysia Broda. 2015. Proof of the soundness and completeness of ilasp2.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. In *The Thirteenth International Conference on Learning Representations*.

Vladimir Lifschitz. 2019. *Answer set programming*, volume 3. Springer Heidelberg.

Peter Linz and Susan H Rodger. 2022. *An introduction to formal languages and automata*. Jones & Bartlett Learning.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. 2024. Grammar-aligned decoding. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

Matthew Renze. 2024. The effect of sampling temperature on problem solving in large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7346–7356, Miami, Florida, USA. Association for Computational Linguistics.

Subhro Roy, Sam Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, and Benjamin Van Durme. 2023. BenchCLAMP: A benchmark for evaluating language models on syntactic and semantic parsing. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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.

Karthik Valmeekam, Kaya Stechly, Atharva Gundawar, and Subbarao Kambhampati. 2025. A systematic evaluation of the planning and scheduling abilities of the reasoning model o1. *Transactions on Machine Learning Research*.

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. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia

Kulikov, and Zaid Harchaoui. 2024. From decoding to meta-generation: Inference-time algorithms for large language models. *arXiv preprint arXiv:2406.16838*.

Brandon T. Willard and Rémi Louf. 2023. Efficient guided generation for large language models. *Preprint*, arXiv:2307.09702.

A ASG Example and Learning Details

A.1 ASG Example

```
start ⟶ as bs cs {
    :- size(X)@1, not size(X)@2.
    :- size(X)@1, not size(X)@3.
}

as ⟶ "a" as {
    size(X+1) :- size(X)@2.
} | {
    size(0).
}

bs ⟶ "b" bs {
    size(X+1) :- size(X)@2.
} | {
    size(0).
}

cs ⟶ "c" cs {
    size(X+1) :- size(X)@2.
} | {
    size(0).
}
```

Figure 2: The learned ASG for $a^n b^n c^n$ using our approach. This grammar utilizes ASP constraints (in bold and surrounded by {}) to enforce the context-sensitive condition that all three symbol sequences maintain equal length.

Figure 2 illustrates the ASG learned via the ASG learner based on ILASP for the language $L = \{a^n b^n c^n : n \geq 1\}$. The ASG consists of two key aspects:

1. A CFG expressed in Extended Backus–Naur form, i.e., as → "a" as. Here, the non-terminals are as, bs, and cs, the terminals are a, b, and c, the start symbol is start, and → denotes the production rules (i.e., the non-terminal on the left-hand side of the arrow can be replaced by the terminal on the right-hand side of the arrow).

2. Context-sensitive constraints annotating the production rules expressed in ASP code (for further details on ASP, please see Lifschitz (2019)). The constraints are encoded via curly braces {...} in the ASG and illustrated in bold text. The first rule's constraints enforce that all three non-terminals must generate sequences of equal length by requiring size(X) to be consistent across all child positions. Terminal productions implement a counting mechanism where each recursive rule increments the size counter by one, while base cases initialize size(0). The @ symbol refers to specific child positions in productions and parse trees, enabling position-dependent constraint checking. For example, size(X)@1 refers to the count accumulated in the first child of the parse tree.

A.2 Constraint Learning via ASG Learner and ILASP.

Section 4.1 provides an intuitive description of how the ASG learner, based on ILASP, learns the context-sensitive constraints. Following (Law et al., 2019), we now formally define an ASG learning task as $T = \langle G_{\text{CFG}}, S_M, \langle E^+, E^- \rangle \rangle$. Here, G_{CFG} serves as the base CFG grammar, S_M is the search space of possible ASP annotations on production rules to construct G_{ASG}, and E^+, E^- are positive and negative examples, respectively.

Given these inputs, ILASP learns a minimal hypothesis $H \subseteq S_M$ containing ASP annotations over G_{CFG} such that:

$$\forall y \in E^+ : y \in L(G_{\text{CFG}} : H) \quad (5)$$

$$\forall y \in E^- : y \notin L(G_{\text{CFG}} : H) \quad (6)$$

where $G_{\text{CFG}} : H$ denotes the ASG (G_{ASG}) constructed by extending G_{CFG} with annotations from H. The learned constraints in H encode context-sensitive rules (e.g., enforcing $\text{count}(a) = \text{count}(b) = \text{count}(c)$ for $a^n b^n c^n$ as in Figure 2). Given ILASP searches for a solution covering all examples, we remove duplicate samples when we feed E^+ and E^- to the ASG learner.

Model	G	Accuracy	
		$a^n b^n c^n$	$a^n b^n c^m$
GPT 4.1	-	63.3%	76.7%
o1	-	86.7%	96.7%
o3 mini	-	63.3%	86.7%
o4 mini	-	90.0%	93.3%
DeepSeek-R1	-	80.0%	86.7%
Llama 1B	-	20.0%	6.7%
Llama 1B	$\mathbf{G_{ASG}}$	**100.0%**	**100.0%**
Llama 1B	$\mathbf{\hat{G}_{ASG}}$	**100.0%**	**100.0%**
Llama 3B	-	20.0%	23.3%
Llama 3B	$\mathbf{G_{ASG}}$	**100.0%**	**100.0%**
Llama 3B	$\mathbf{\hat{G}_{ASG}}$	**100.0%**	**100.0%**
Llama 8B	-	46.7%	10.0%
Llama 8B	$\mathbf{G_{ASG}}$	**100.0%**	**100.0%**
Llama 8B	$\mathbf{\hat{G}_{ASG}}$	**100.0%**	**100.0%**
Llama 70B	-	76.7%	53.3%
Llama 70B	$\mathbf{G_{ASG}}$	**100.0%**	**100.0%**
Llama 70B	$\mathbf{\hat{G}_{ASG}}$	**100.0%**	**100.0%**

Table 2: Accuracy results for $a^n b^n c^n$ and $a^n b^n c^m$ with different LLMs (Model), including Llama 3.2 3B and 3.1 8B, and grammar constraints (G).

B Further Results

Section 5.3 showcased context-sensitive accuracy results with respect to various LLMs and grammar constraints. Here, Table 2 presents results with Llama 3.2 3B and Llama 3.1 8B, which we omitted from the main text due to space requirements. Similar conclusions can be drawn as before. Hence, we omit any further discussions.

C GPU Specification

Our experiments were conducted using a GPU cluster with nodes containing 2× Intel Xeon Platinum 8358 CPUs (2.60GHz, 32 cores each) and NVIDIA L40S GPUs (48GB GDDR6), where we utilized up to 4 GPUs with 96GB RAM.

D Prompt Example

System Instruction:

You are an expert in formal languages, specifically, Context-Free and Context-Sensitive Grammars. You can read and understand grammars, and given a grammar specification, you can generate words that consistently conform to the grammar, its language, and rules without a single mistake. For each message, generate a word (a sequence of characters belonging to the language) that conforms to the grammar specification $a^nb^nc^n$. This grammar represents the language of strings consisting of n number of a's, followed by n number of b's, and finally followed by n number of c's, where all n's are equal (i.e., the number of a's, b's, and c's are all equal) and in the specified order. Each message will specify a max n value, meaning, the individual number of a's, b's, and c's cannot exceed that amount, and you must aim to maximise n (length of words) up to the specified max, thereby, prefering longer words of the grammar's language.

Only respond with a single word that conforms to the grammar, do not generate any additional text beyond the correct word with respect to the grammar.

Example Interactions:

User: Generate a valid word/string of the grammar $a^nb^nc^n$, where you should prefer larger numbers of n (i.e., longer sequences) and the max n value is 3.
Assistant: `aaabbbccc`

User: Generate a valid word/string of the grammar $a^nb^nc^n$, where you should prefer larger numbers of n (i.e., longer sequences) and the max n value is 10.
Assistant: `aaaaaaaaaabbbbbbbbbbcccccccccc`

Figure 3: Prompt template for the $a^nb^nc^n$ language generation task. The system instruction defines the formal language requirements, followed by example interactions demonstrating expected inputs and outputs.

Figure 3 illustrates the prompt used for the $a^nb^nc^n$ task, with a similar style for our $a^nb^nc^m$ task. Akin to Albinhassan et al. (2025), we adopt a standard few-shot prompting strategy, where we provide a description of the task, syntax, and constraints in natural language and formal language notation.