GRAMMAR-LLM: Grammar-Constrained Natural Language Generation

Gabriele Tuccio[1,2], Luana Bulla[1,2], Maria Madonia[1], Aldo Gangemi[2,3], Misael Mongiovì[1,2],

[1]University of Catania, Italy, [2]ISTC - National Research Council, Italy , [3]University of Bologna, Italy

Correspondence: misael.mongiovi@unict.it

Abstract

Large Language Models have achieved impressive performance across various natural language generation tasks. However, their lack of a reliable control mechanism limits their effectiveness in applications that require strict adherence to predefined taxonomies, syntactic structures, or domain-specific rules. Existing approaches, such as fine-tuning and prompting, remain insufficient to ensure compliance with these requirements, particularly in low-resource scenarios and structured text generation tasks.

To address these limitations, we introduce GRAMMAR-LLM, a novel framework that integrates formal grammatical constraints into the LLM decoding process. GRAMMAR-LLM enforces syntactic correctness in linear time while maintaining expressiveness in grammar rule definition. To achieve this, we define a class of grammars, called LL(prefix), – which we show to be equivalent to LL(1) – specifically designed for their use with LLMs. These grammars are expressive enough to support common tasks such as hierarchical classification, vocabulary restriction, and structured parsing. We formally prove that LL(prefix) grammars can be transformed into LL(1) grammars in linear time, ensuring efficient processing via deterministic pushdown automata. We evaluate GRAMMAR-LLM across diverse NLP tasks, including hierarchical classification, sign language translation, and semantic parsing. Our experiments, conducted on models such as LLaMA 3 (for classification and translation) and AMRBART (for parsing), demonstrate that GRAMMAR-LLM consistently improves task performance across zero-shot, few-shot, and fine-tuned settings[1].

1 Introduction

Large Language Models (LLMs) have shown remarkable performance across a wide range of natural language generation tasks, including machine translation, text summarization, and open-domain text generation (Naveed et al., 2023). Despite these advancements, generative models inherently lack constraints that align their outputs with predefined taxonomies (Geng et al., 2025). Many real-world applications necessitate fine-grained control over generated text, such as enforcing specific syntactic structures or adhering to domain-specific constraints. Consequently, these models often prove inadequate for tasks that demand strict lexical and structural conformity, such as classification and question answering over predefined choices. Furthermore, given that LLMs are highly dependent on the data seen during their training phase, they exhibit significant performance degradation when tasked with generating text for languages or modalities that fall outside the distribution of their training data. This limitation is particularly evident when generating text in low-resource scenarios, such as for under-represented languages (e.g., sign language or ancient languages), where insufficient training data hinders the model's ability to learn necessary structures. Ensuring that Language Models (LMs) adhere to predefined structural and lexical constraints remains a significant challenge.

To address these challenges, we introduce GRAMMAR-LLM, a novel approach that integrates formal grammars directly into the LLM decoding process (Fig. 1). Rather than relying on post-hoc validation or heuristic filtering, GRAMMAR-LLM treats text generation as an automaton-driven process, ensuring that the generated output adheres to a predefined Context-Free Grammar (CFG) (Hopcroft et al., 2001)

Integrating grammatical constraints into LLMs introduces several challenges, primarily due to the intrinsic complexity of formal grammars and the trade-off between expressiveness and computational efficiency. While highly expressive grammars provide greater control over text generation,

[1]the source code and other materials are available at: `https://github.com/misaelmongiovi/grammarllm`

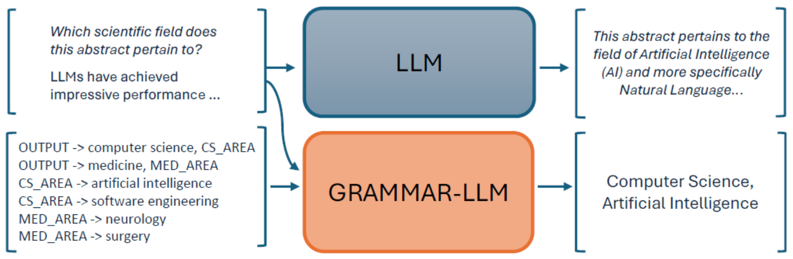
Which scientific field does this abstract pertain to?
LLMs have achieved impressive performance ...
OUTPUT -> computer science, CS_AREA
OUTPUT -> medicine, MED_AREA
CS_AREA -> artificial intelligence
CS_AREA -> software engineering
MED_AREA -> neurology
MED_AREA -> surgery
LLM
GRAMMAR-LLM
This abstract pertains to the field of Artificial Intelligence (AI) and more specifically Natural Language...
Computer Science, Artificial Intelligence

Figure 1: GRAMMAR-LLM enforces LLMs to adhere to a predefined grammar, thus avoiding verbose answers when a structured output is required

they often introduce computational intractability when applied sequentially in an autoregressive manner. An alternative is to restrict the system to grammars that can be processed in linear time, such as LL(1) (Aho et al., 2006). However, constructing an LL(1) grammar that treats tokens as terminals is extremely challenging—especially when using subword tokenization, as is common with LLMs—since token sequences on the right-hand side of rules may violate LL(1) constraints. To mitigate this, we define a novel notation for traditional LL(1) grammars – which are efficiently parsable in a left-to-right manner – called LL(prefix), which enables users to define grammars without delving into the details of tokenization. We then constrain the model to generate only tokens that comply with a given LL(prefix) grammar by leveraging a Push-Down Automaton (PDA) (Deutsch et al., 2019) automatically derived from the input grammar. Our approach is adaptable to different LM architectures, including GPT-based and BART-based models, and guarantees that the generated output strictly conforms to predefined grammatical structures.

To assess the performance of our method across different Natural Language Processing (NLP) tasks, we focus on three key domains: hierarchical classification (Silla and Freitas, 2011), sign language translation (Núñez-Marcos et al., 2023), and semantic parsing (Kamath and Das, 2018). Each of these tasks presents challenges in generating structured outputs that conform to predefined taxonomies or specialized lexicons, which can be effectively represented using a CFG. Specifically, hierarchical classification requires the generation of outputs that accurately preserve hierarchical relationships, ensuring the validity of generated taxonomy paths. Sign language translation presents the additional challenge of bridging the modality gap between sign notation and natural language, particularly in data-scarce scenarios. Meanwhile, semantic parsing demands strict adherence to predefined semantic schemas. To assess the impact of different model architectures and parameter scales on structured text generation, we test 1B, 8B, and 70B variants of the LLaMA-3 model (Dubey et al., 2024), as well as a BART-based model (Bai et al., 2022). Across all tasks and learning settings, GRAMMAR-LLM consistently outperforms baseline models, yielding substantial improvements in classification accuracy, translation quality, and adherence to structured constraints. These findings highlight the effectiveness of integrating formal grammars into language models, particularly in scenarios that demand structured and highly precise output.

The main contributions of this work are:

- We present GRAMMAR-LLM, a novel framework that integrates formal grammars into the generation process of language models. This approach ensures that generated output strictly adhere to predefined syntactic and structural requirements, addressing the limitations of existing methods in enforcing fine-grained control over text generation.
- We propose LL(prefix), a novel formalization that generalizes the LL(1) class of CFG grammars enabling the user to define grammars without delving into the details of LLM tokenizers, thus enhancing expressiveness without sacrificing efficiency.
- We present an algorithm that turns an LL(prefix) grammar into an LL(1) grammar, therefore showing theoretical equivalence and enabling real-time, token-by-token guidance during autoregressive generation.
- We demonstrate the adaptability of GRAMMAR-LLM across a variety of tasks (namely hierarchical classification, sign

language translation, and AMR translation) and model architectures (LLaMA-3, AMR-BART). Our method consistently improves performance in zero-shot, one-shot, few-shot, and fine-tuned settings, highlighting its broad applicability.

The paper is structured as follows: Section 2 outlines related work. In Section 3, we introduce related background literature and describe our GRAMMAR-LLM framework. Section 4 presents an experimental analysis highlighting the gain in performance introduced by the proposed approach. Finally, Section 5 and 6 conclude the paper and discuss limitations.

2 Related Works

Recently, several Grammar-Constrained Decoding (GCD) frameworks (Geng et al., 2023; Koo et al., 2024; Li et al., 2024; Park et al., 2024, 2025; Scholak et al., 2021; Shin et al., 2021; Willard and Louf, 2023; Deutsch et al., 2019) have been developed to constrain decoding. These methods primarily leverage finite state automata, CFGs, and deterministic parsing strategies to enforce structured output constraints.

Koo et al. (2024) and Park et al. (2025) leverage automata-based methods to enforce structural compliance while improving computational efficiency. Koo et al. employ finite-state transducers (FSTs) alongside Finite State Machines (FSMs), providing a closed-form solution for regular languages and extending to deterministic context-free languages via PDAs. Their method achieves a substantial speedup in constraint enforcement and enables modular adaptation across structured generation tasks. Park et al. refine this approach by introducing a token spanner table, which efficiently maps LLM tokens to sequences of grammar terminals, optimizing both offline preprocessing and online constrained decoding.

Li et al. (2024) propose Formal-LLM, which constrains plan generation by non-deterministic PDAs. The approach enables building PDAs directly from natural language and integrating transitions directly into LLM prompts to guide the text generation. This approach does not intervene in the decoding process, where specific conditions are required.

Willard and Louf (2023) propose Efficient Guided Generation, which models text generation as a sequence of FSM state transitions, enabling constraints enforcement with O(1) complexity per token. While computationally efficient, this approach is inherently limited to regular languages, restricting its applicability to more complex structures. Geng et al. (2023), PICARD (Scholak et al., 2021), and Shin et al. (2021) instead employ grammar-based decoding methods that rely on input-dependent grammars (IDGs) or external parsing mechanisms for structured text generation, particularly in semantic parsing.

Park et al. (2024) introduce Grammar-Aligned Decoding (GAD), which tends to preserve the LLM's distribution among valid outputs, and employ the formal framework proposed by Geng et al. (2023), making it complementary to our approach. Similarly, Ahmed et al. (2024) employ logic circuits to express constraints and locally constrained resampling to enforce constraints while maintaining expressiveness, correcting biased samples through importance weighting and resampling. Zhang et al. (2024, 2023) integrate Hidden Markov Models (HMMs) to impose logical constraints on LLM outputs, using probabilistic reasoning to estimate token validity and efficiently compute conditional probabilities during the generation process.

Unlike FSM-based methods, GRAMMAR-LLM supports a wider class of context free grammars. Unlike probabilistic and external parsing approaches, it guarantees strict adherence to grammar rules without requiring complex data structures, inefficient superlinear processing or post-filtering mechanisms. This makes GRAMMAR-LLM a more scalable and adaptable solution for constrained text generation.

Deutsch et al. (2019) present a general-purpose algorithm for constrained sequential inference, which integrates constraints into the decoding process using deterministic pushdown automata. They consider the automata as an input and do not define the class of grammars which can be efficiently handled by their approach.

Dong et al. (2024) present XGrammar, a structured generation engine for LLM that enables efficient and flexible constrained decoding using context-free grammars. By partitioning the vocabulary into context-independent tokens, XGrammar reduces the computational overhead typically associated with grammar-based decoding. However, XGrammar does not guarantee linear-time overhead, as it relies on a non-deterministic PDA to manage grammar ambiguities. This non-determinism necessitates maintaining multiple stacks – which may be intractable – to handle grammar ambiguities.

GRAMMAR-LLM enforces constraints directly at the decoding level using deterministic pushdown automata, which are automatically constructed from LL(prefix) grammars, ensuring that generated outputs strictly adhere to predefined syntactic structures. Unlike more general approaches, GRAMMAR-LLM ensures efficiency by providing a theoretical guarantee that the overhead grows linearly with the length of the generated text.

3 A Novel Framework for Grammar-Constrained LLM Decoding

Our method builds upon the literature on context-free grammars to design an efficient mechanism that enforces LLMs to generate grammar-compliant output. We first introduce the literature on LL(1) grammars and their implementation by means of pushdown automata, which enable left-to-right processing in linear time and are therefore suitable for integration with LLMs. Then, we introduce a novel and more expressive formalization, named LL(prefix), which is well-suited for representing sequences of tokens in a user-friendly way, and give an algorithm to transform LL(prefix) grammars into LL(1) grammars. Eventually, we show how to integrate a deterministic pushdown automaton with a transformer decoder to enforce text generation that adheres to the input LL(prefix) grammar. An overview of our pipeline is shown in Figure 2.

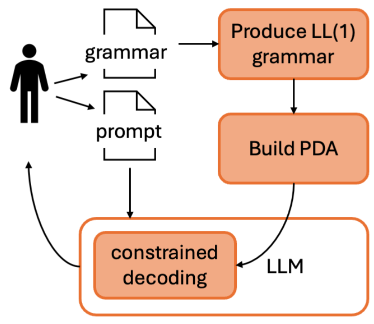

Figure 2: Overview of our GRAMMAR-LLM pipeline.

3.1 Background

In this section we briefly introduce LL(1) grammars and its use for verification by means of pushdown automata.

We begin with the definitions of CFGs and LL(1) grammars (Hopcroft et al., 2001).

Definition 1 *A grammar $G = (\Sigma, N, P, S)$ is context-free if all its production are of the form $A \to \alpha$ with $A \in N$ and $\alpha \in (\Sigma \cup N)^*$.*

A significant subclass of the family of context-free grammars is the set of LL(1) grammars. The definition of LL(1) grammar is based on two functions $FIRST$ and $FOLLOW$ associated with the grammar. For any $\alpha \in (\Sigma \cup N)^*$, $FIRST(\alpha)$ is the set of terminals that begin the strings derived from α. If $\alpha \to \epsilon$ then $\epsilon \in FIRST(\alpha)$.

For any $A \in N$, $FOLLOW(A)$ is the set of terminals a that can appear immediately to the right of A in some sentential form (i.e. the set of terminals a such that there exists a derivation of the form $S \Rightarrow^* \alpha A a \beta$). If A can be the rightmost symbol in some sentential form, the \$ is in $FOLLOW(A)$, where \$ is an endmarker symbol.

Definition 2 *A context-free grammar $G = (\Sigma, N, P, S)$ is LL(1) if whenever $A \to \alpha \mid \beta$ are two distinct productions of G, the following conditions hold:*

1) For no terminal $a \in \Sigma$, both α and β derive a string beginning with a (i.e. no terminal $a \in \Sigma$ is in $FIRST(\alpha) \cap FIRST(\beta)$)

2) At most one of α and β can derive ϵ

3) If $\beta \Rightarrow^* \epsilon$, then α does not derive any string beginning with a terminal in $FOLLOW(A)$ (i.e. if $\alpha \Rightarrow^* a\alpha_1$ then $a \notin FOLLOW(A)$).

Languages accepted by LL(1) grammars can be recognized using pushdown automata (PDAs). We now give a formal definition of a PDA, followed by a description of the algorithm to construct the automaton.

A *pushdown automaton*, PDA, for short, is a system $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where Q is a finite set of *states*, Σ is the *input alphabet*, Γ is the *stack alphabet*, $q_0 \in Q$ is the *initial state*, $Z_0 \in \Gamma$ is the *start symbol*, $F \subseteq Q$ is the set of *final sates* and δ, the *transition function*, is a mapping from $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.

The interpretation of $\delta(q, a, Z) = \{(p_1, \gamma_1), \dots (p_m, \gamma_m)\}$, with $q, p_1 \dots p_m \in Q$, $a \in \Sigma$, $Z \in \Gamma$ and $\gamma_1 \dots \gamma_m \in \Gamma^*$, is that the PDA in the state q, with input symbol a and symbol Z at the top of the stack, can for any i enter state p_i, replace symbol Z by γ_i and advance the input head one symbol. The interpretation of $\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), \dots (p_m, \gamma_m)\}$ is that the PDA in the state q, independent of the input symbol being scanned and symbol Z at the top of the stack, can enter state p_i, replace symbol Z by

γ_i. In this case the input head is not advanced. For a PDA the language accepted by empty stack is defined in terms of *instantaneous descriptions*, ID for short, that formally describe the configurations of a PDA at a given instant: an ID is a triple (q, w, γ) where $q \in Q$ is the current state, $w \in \Sigma$ is the "unexpended input" and $\gamma \in \Gamma^*$ is the stack content. Given two ID, I_1 and I_2, we write $I_1 \vdash^* I_2$ if I_2 can be reached from I_1 after 0 or more steps. Then, for a PDA M, the language accepted by empty stack is $N(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$ for some $p \in Q\}$. A PDA is said *deterministic* if at most one move is possible from any ID.

It is known, that any language generated by a context-free grammar can be accepted, by empty stack, by a PDA M (Hopcroft et al., 2001); but, here, since we consider LL(1) context-free grammars and since we need deterministic PDA, we present a different algorithm for the construction of M (see Algorithm 1) inspired by the construction of a Parsing Table for an LL(1) grammar (Aho et al., 2006).

Algorithm 1 Construction of a PDA, M, from an LL(1) grammar G

```
Require: An LL(1) grammar $G = (\Sigma, N, P, S)$
Ensure: A pushdown automaton $M = (Q, \Sigma \cup \{\$\}, \Gamma, \delta, q_0, Z_0, \emptyset)$ equivalent to $G$
 1: init. $Q \leftarrow \{q_a \mid a \in \Sigma\} \cup \{q_0, q_\$\}$
 2: init. $\Gamma \leftarrow \Sigma \cup N \cup \{\$\}$
 3: init. $Z_0 \leftarrow \$S$
 4: For any $a \in \Sigma \cup \{\$\}$ and $Z \in N \cup \{\$\}$ do $\delta(q_0, a, Z) \leftarrow (q_a, Z)$
 5: For any $a \in \Sigma \cup \{\$\}$ do $\delta(q_a, \epsilon, a) \leftarrow (q_0, \epsilon)$
 6: for all productions $A \to \alpha$ in $P$ do
 7:     For each terminal $a \in FIRST(\alpha)$ $\delta(q_a, \epsilon, A) \leftarrow (q_a, \alpha)$
 8:     If $\epsilon \in FIRST(\alpha)$, for any $b \in FOLLOW(A)$, $\delta(q_b, \epsilon, A) \leftarrow (q_b, \alpha)$
 9:     If $\epsilon \in FIRST(\alpha)$ and $\$ \in FOLLOW(A)$, $\delta(q_\$, \epsilon, A) \leftarrow (q_\$, \alpha)$
10: end for
11: Return $M$
```

Theorem 1 *Algorithm 1, with an LL(1) grammar G in input, produces in linear time a deterministic PDA, M, that is equivalent to G, i.e. $N(M) = L(G)\$$, where $L(G)\$$ is the set of strings in $L(G)$ followed by an endmarker \$.*

Note that, since G is LL(1) then $|\delta(q, a, A)| \leq 1$, for any $a \in \Sigma \cup \{\$, \epsilon\}$ and $A \in \Gamma$. Moreover, if $\delta(q, \epsilon, A) \neq \emptyset$, then $\delta(q, a, A) = \emptyset$, for any $a \in \Sigma \cup \{\$\}$. Hence M is a deterministic PDA. At last, one can show, that $N(M) = L(G)\$$.

3.2 A Novel Expressive Grammar Notation for Constraining LLM Output

Despite implementable with pushdown automata, LL(1) grammars exhibit limited expressiveness when applied to LLMs. This limitation arises because grammars are traditionally defined in terms of symbols, which can correspond to words or characters. However, the fundamental semantic unit for LLMs is the token, which often represents subword segments rather than entire words. Using tokens as terminal symbols in a grammar would make defining rules cumbersome for users, making this approach impractical. While words or entire sentences in a grammar can be easily converted into sequences of tokens, this conversion may violate the constraints of an LL(1) grammar. To illustrate this issue, consider the following grammar:

$S \to$ uncertain S | undefined S | ϵ

If we consider the words "uncertain" and "undefined" as terminals, this grammar is LL(1). However if we adopt subword tokenization where the prefix "un" is separated by the rest of the word, the grammar turns into:

$S \to$ un certain S | un defined S | ϵ

which is violates condition 1) of LL(1) grammars (Def. 2) since the RHS of two different rules with the same LHS begin with the same terminal "un".

To overcome this limitation, we define a class of grammars, – which we show to be equivalent in generative capacity to LL(1) – named $LL(prefix)$, which allows multiple rules to share sequences of symbols in the initial portion of the RHS of production rules. The definition generalizes traditional $LL(1)$ grammars by accommodating prefix-sharing structures. We then demonstrate that any $LL(prefix)$ grammar can be transformed into an equivalent $LL(1)$ grammar in linear time and space (Theorem 2 below). This transformation ensures that $LL(prefix)$ grammars remain efficiently parseable and can be verified using pushdown automata, maintaining integrability with LLMs, as we show in Sect. 3.3.

Definition 3 *A context-free grammar $G = (\Sigma, N, P, S)$ is LL(prefix) if whenever $A \to \omega\ \alpha \mid \omega\ \beta$ are two distinct productions of G with $\omega \in \Sigma^*$ and α and β starting with a different symbol, the following conditions hold:*

1) For no terminal $a \in \Sigma$, both α and β derive a string beginning with a (i.e. no terminal $a \in \Sigma$ is in $FIRST(\alpha) \cap FIRST(\beta)$)

2) At most one of α and β can derive ϵ
3) If $\beta \Rightarrow^* \epsilon$, then α does not derive any string beginning with a terminal in $FOLLOW(A)$ (i.e. if $\alpha \Rightarrow^* a\alpha_1$ then $a \notin FOLLOW(A)$).

In short, with respect to LL(1) grammars, Definition 3 admits prefixes of terminals of any length to be shared among different rules with the same LHS.

We give an algorithm to transform an LL(prefix) grammar into an LL(1) grammar and prove that it correctly generates an equivalent LL(1) grammar in linear time and space.

Algorithm 2 Transforming an LL(prefix) grammar into an LL(1) grammar

```
Require: An LL(prefix) grammar $G = (\Sigma, N, P, S)$
Ensure: An equivalent LL(1) grammar $G'$
 1: while there are pairs of productions of the type $A \to a\ \alpha$
    and $A \to a\ \beta$ for some $A \in N$ and $a \in \Sigma$ do
 2:     Take all productions of the kind $A \to a\ \alpha_i$ for every
        $\alpha_i \in (\Sigma \cup N)^*$ and substitute them with: $A \to a\ B$
        and $B \to \alpha_i$ for every $\alpha_i$, where $B$ is a newly generated non-terminal
 3:     Add $B$ to $N$
 4: end while
 5: Return $G' = (\Sigma, N, P, S)$
```

Theorem 2 *Algorithm 2 produces an LL(1) grammar G' that is equivalent to the input LL(prefix) grammar G with time and space complexity $O(n \cdot m)$ where n is the number of production rules and m is the maximum length of production rules.*

The proof of the theorem is available in the supplemental material.

3.3 Grammatically-Compliant Text Generation

We show how to use a pushdown automaton to enforce a Transformer decoder to generate valid text conforming to an LL(prefix) grammar. As described in Section 3.2, any LL(prefix) grammar can be transformed into an equivalent LL(1) grammar. In turn, an LL(1) grammar can be recognized by a pushdown automaton, as discussed in Section 3.1. In the following, we describe how to integrate a pushdown automaton into the generation process of an LLM to ensure grammar compliance.

LLMs generate tokens one by one following an autoregressive decoding approach, where each new token is predicted based on input and previously generated tokens. To do so, the output from the last decoder layer is transformed to the vocabulary space using a linear projection and a softmax function:

$$\begin{aligned} P(y_t \mid x_1, \ldots, x_n, y_1, \ldots, y_{t-1}) = \\ = \mathrm{softmax}(W_o H_t + b_o) \end{aligned} \tag{1}$$

where H_t is the output from the last decoder layer and W_o, b_o are learnable parameters. The process repeats iteratively until a stopping criterion is met.

We show that a pushdown automaton can be integrated with the decoding process to force the LLM to follow the input grammar. Our automaton considers tokens as terminal symbols. The generation process is described in Algorithm 3.

Algorithm 3 Autoregressive Token Generation

```
Require: Initial sequence of tokens $X = \{x_1, x_2, \ldots, x_n\}$,
    pushdown automaton $M = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$,
    model parameters $\theta$
Ensure: Generated token sequence $Y = \{y_1, y_2, \ldots, y_m\}$
 1: init. $Y \leftarrow X$
 2: init. PDA configuration $(q, \gamma) \leftarrow (q_0, \gamma_0)$
 3: while $\gamma$ is not empty do
 4:     $H_t \leftarrow$ Transformer$(Y, \theta)$
 5:     $P(y_t \mid Y) \leftarrow \mathrm{softmax}(W_o H_t + b_o) \circ$
        next_terminals$((q, \gamma))$
 6:     sample next token $y_t$ and append it to $Y$
 7:     do transition $(q, \gamma) \leftarrow \delta(q, y_t, \mathrm{top}(\gamma))$
 8:     while $q \neq q_0$ do
 9:         do transition $(q, \gamma) \leftarrow \delta(q, \epsilon, \mathrm{top}(\gamma))$
10:     end while
11: end while
12: return $Y$
```

The differences with respect to standard decoding are in lines 2,3,5 and 7-10. Line 2 initializes the automaton. The exit condition in Line 3 is actually equivalent to standard decoding since the stack is empty when the end of sentence symbol \$ is generated. Line 5 generates the probability distribution for the next token considering forbidden tokens. next_terminals(q, γ) generates the set of valid tokens given the current automaton configuration and return them as a one-hot vector. The operator $\circ$ performs the element-wise product between two vectors. For LL(1) grammars, next_terminals$((q, \gamma))$ simply return $FIRST(\gamma)$. Lines 7-10 perform automaton transitions based on its current state, the newly generated token and the top of the stack. The first transition consumes the generated token and moves the automaton to a token-specific state. Transitions are performed until the automaton returns to the initial state, i.e. it is ready to accept another token.

4 Results and Evaluation

To assess the effectiveness of our method, we conduct experiments across three distinct NLP tasks (i.e. hierarchical classification, sign language translation and semantic parsing), each chosen to test different aspects of constrained text generation. Specifically, in the hierarchical classification task (Silla and Freitas, 2011), we assess the model's capability to generate text that conforms to predefined taxonomies, a critical requirement in structured data generation and classification scenarios. Sign language translation (Núñez-Marcos et al., 2023) refers to the translation from natural language to glosses, where sign language is encoded using gloss notation. This notation aligns signs with words or phrases from a spoken language, serving as a structured, text-based annotation rather than a direct translation. This task poses a unique challenge due to the inherent modality gap between natural language and gloss-based representations, as well as the scarcity of high-quality parallel data. We employ this case study to highlight the effectiveness of GRAMMAR-LLM in low-resource and modality-specific contexts, where conventional LLMs often struggle with distributional mismatches. The semantic parsing (Kamath and Das, 2018) demands specific alignment between generated text and predefined semantic schemas, making it a robust benchmark for assessing GRAMMAR-LLM's ability to enforce formal grammatical constraints. This task underscores the model's adaptability to fine-tuned structured generation tasks, ensuring that outputs maintain both semantic accuracy and syntactic coherence[2].

Table 1 presents the results for the hierarchical classification task in terms of micro F1 score. Additionally, we report the percentage of outputs that conform to the requirements to highlight the impact of constraint enforcement. We test our method on the Web of Science (WoS) dataset (Kowsari et al., 2017), which consists of approximately 50,000 research abstracts annotated with a two-level taxonomy consisting of 7 and 134 labels at Levels 1 and 2, respectively. For evaluation, we select 2,000 instances using a stratified sampling strategy at Level 1 to ensure proportional representation, while employing random sampling at Level 2. In all cases, the prompt explicitly specifies the full taxonomy and the expected output format. We assess the performance of the LLaMA-3[3] model in three configurations (1B, 8B, and 70B) across zero-shot, one-shot, and few-shot scenarios. In the few-shot setting, we provide 10 examples, selected using the same sampling criteria as the test set. As a baseline, we compare these models with and without our GRAMMAR-LLM (Sect. 3) approach (which we name G-LlaMa) to quantify its impact on classification performance. We employ a simple grammar that forces the model to first generate a Level 1 class, then generate a Level 2 class that is compatible with the Level 1 class.

As shown in Table 1, G-LLaMA outperforms unconstrained LLaMA models across all experimental settings. The G-LLaMa model achieves the best results in the zero-shot scenario, with F1 scores of 0.734 and 0.504 at the first and second taxonomy levels, respectively. An analysis of Level 1 classification reveals that the most significant improvements occur in the smallest model (i.e. LLaMA-1B). In this case, the constrained model achieves an F1 score improvement of 30 and 28 percentage points over the unconstrained baseline in the zero-shot and one-shot configurations, respectively, showing comparable performance in the few-shot settings. These findings suggest that larger models with more parameters benefit less from the constraints at the first taxonomy level, as it is easier to classify due to broader, distinct categories. In contrast, smaller models benefit more from the grammar-based module, which enables them to generate valid, concise outputs and improves accuracy, as demonstrated by LLaMA-1B's zero-shot and one-shot performance. At a more granular taxonomy level (i.e. Level 2), the constrained models exhibit more pronounced improvements over the baseline models across all configuration settings. Specifically, G-LlaMa-1B outperforms the unconstrained model by 12, 16, and 8 percentage points in the zero-shot, one-shot, and few-shot settings, respectively. A similar trend is observed for LLaMA-8B and LLaMA-70B, where the grammar-based models yield an F1 score improvement of 5, 4, and 6 percentage points for the Llama-8B, and 4, 2, and 2 points for Llama-70B, respectively, across the zero-, one- and few-shot settings. At the

[2]Further details on the prompts used in the experiments (Tables 1 and Table 3) and the grammars implemented for the three case studies are provided in the supplementary materials.

[3]We chose LLama-3 as a representative LLM since it is one of the most popular open-source model and it achieves competitive performances with respect to commercial models. Most commercial models (e.g. GPT-4) cannot be employed since we cannot intervene in their decoding process.

second-level taxonomy, the grammar-based constraints lead to more significant improvements, mitigating the generative models' verbosity. This ensures adherence to a valid hierarchical path and avoids the generation of invalid taxonomy labels[4].

Notably, the percentage of outputs conform to the requirements (i.e. referred to as validity in Table) closely follows the improvements in F1 score. The constrained models consistently achieve a 100% validity rate across all configurations, which indicate outputs that fully comply with the target taxonomy. In contrast, unconstrained models yield substantially lower validity rates, with performance varying significantly across model sizes. Specifically, LLaMA-1B, LLaMA-8B, and LLaMA-70B achieve best validity rates of 27%, 69%, and 85%, respectively, showing an improvement as the number of examples in the prompt increases. Nevertheless, none of the unconstrained models reach full validity, adversely affecting their overall performance.

To assess the types of errors that our approach mitigates, Table 2 presents the main error categories produced by unconstrained models in the classification task across all prompt configurations. We classify three mutually exclusive main categories of invalid outputs: invalid taxonomy labels (i.e. cases where a label is appropriate at a general level but incorrectly assigned at a more specific level); incorrect number of labels (i.e. outputs with excessive hierarchical levels w.r.t. the predefined taxonomy, while remaining accurate at the first two levels), and others (i.e. verbose or free-form responses that do not adhere to the expected classification format). As shown in Table 2, the most frequent error across all models involve the inclusion of invalid taxonomy labels or the omission of required ones. The only exception is observed in LLaMA-1B under the zero- and one-shot settings, where non-conforming outputs predominate. This depends on model's tendency towards verbosity in the absence of sufficient instructional context, often producing too long responses or generating content that is not related to the task. However, in the few-shot configuration, LLaMA-1B aligns with the error profile of larger models. The inclusion of additional examples in the prompt offers more explicit guidance, facilitating a deeper understanding of the task structure and substantially reducing the occurrence of unstructured outputs. LLaMA-8B and LLaMA-70B exhibit invalid taxonomy labels as the primary error, with LLaMA-70B producing virtually no other error category.

[4]As shown in Table 1, LLaMA-70B performs best in the zero-shot setting, likely due to its extensive parameterization. In contrast, few-shot prompting may limit generalization by overly anchoring the model to provided examples

Model	Zero-shot			One-shot			Few-shot		
	L1	L2	Validity	L1	L2	Validity	L1	L2	Validity
LlaMa-1B	.002	.001	0%	.313	.071	16%	.501	.092	27%
G-LlaMa-1B	**.308**	**.124**	100%	**.539**	**.231**	100%	**.504**	**.173**	100%
LlaMa-8B	.569	.229	62%	**.650**	.258	69%	**.680**	.255	62%
G-LlaMa-8B	**.575**	**.282**	100%	**.650**	**.298**	100%	**.680**	**.310**	100%
LlaMa-70B	.720	.464	80%	.740	.437	83%	.700	.362	85%
G-LlaMa-70B	**.734**	**.504**	100%	**.742**	**.456**	100%	**.702**	**.378**	100%

Table 1: Performance on hierarchical classification using the WoS dataset, evaluated in terms of micro F1 score for Level 1 (L1) and Level 2 (L2) classification, as well as the overall validity rate (%) for each model and configuration.

Table 3 provides an analysis of the performance of the constrained and unconstrained LlaMa models at varying model size (i.e. 1B, 8B and 70B) for the sign language translation task in a few-shot setting. We test models on 2,000 randomly sampled instances from the Synthetic English-ASL Gloss Parallel Corpus (Othman and Jemni, 2012), which comprises 87,710 text-gloss pairs, generated through the application of syntactic transformation rules to English text. For each input sentence, we prompt models in few-shot by selecting relevant examples based on cosine similarity (Li and Li, 2023). Specifically, we retrieve the 30 most similar text-gloss pairs and the 50 most similar glosses from the training set. This approach ensures that the prompt remains contextually relevant to the input. For our method we adopt a simple grammar that enforces producing sequences of entries from the set of admissible glosses. We present results in terms of BLEU (Papineni et al., 2002), ChrF (Popović, 2015), F1 score. In addition, we report the number of valid outputs in terms of percentage as in Table 1. BLEU measures the precision of n-grams between the predicted and reference translations, while ChrF focuses on character level to evaluate more granular alignment. The F1 score reflects the balance between precision and recall over the gloss vocabulary, treating the task as multi-label classification and hence discarding the order of glosses.

As shown in Table 3, model performance increases consistently with model size. This effect is more pronounced for smaller models (i.e. 1B and 8B), where the need for improvement is higher. We observe performance gains of 14, 5 and 4 percentage points in terms of F1 score for the 1B, 8B and 70B models, respectively, w.r.t. LlaMa base-

Model	Zero-shot			One-shot			Few-shot		
	Incorrect number of labels	Invalid taxonomy labels	Others	Incorrect number of labels	Invalid taxonomy labels	Others	Incorrect number of labels	Invalid taxonomy labels	Others
LlaMa 1B	0%	0%	100%	2%	30%	53%	0%	70%	3%
LlaMa 8B	2%	23%	13%	0%	28%	3%	0%	38%	0%
LlaMa 70B	0%	17%	2%	0%	17%	0%	0%	14%	0%

Table 2: Error analysis of unconstrained models across all prompt configurations in the classification task.

Model	BLEU	ChrF	F1	Validity
LlaMa-1B	0.31	0.72	0.65	9%
G-LlaMa-1B	**0.47**	**0.76**	**0.79**	100%
LlaMa-8B	0.70	0.90	0.89	39%
G-LlaMa-8B	**0.81**	**0.93**	**0.94**	100%
LlaMa-70B	0.79	0.93	0.92	49%
G-LlaMa-70B	**0.86**	**0.95**	**0.96**	100%

Table 3: Performance of the models on the text-to-gloss ASL translation task, evaluated on the Synthetic English-ASL Gloss Parallel Corpus in terms of BLEU, ChrF, and F1 scores. We report the overall validity rate (%) for each model and configuration.

lines. These findings are consistent with the results reported for the classification task (cf. Table 1). We observe comparable trends in terms of ChrF2 and BLEU metrics. The analysis of output validity mirrors these trends, with all constrained models achieving 100% output validity, in contrast to significantly lower rates for their unconstrained counterparts. This discrepancy depends on the tendency of unconstained models to generate non-existent glosses, that is not possible with the structural constraints imposed in G-LLaMA models.

These findings highlight the impact of GRAMMAR-LLM in enhancing model performance, especially in translation tasks involving limited data and underrepresented domains. Specifically, the grammatical module contributes to selecting glosses that consistently align with the vocabulary, thereby avoiding errors due to the generation of invalid labels.

Table 4 evaluates the performance of constrained and unconstrained BART-based models (AMR-BART-base - 139M - and AMR-BART-large - 406M (Bai et al., 2022)) for Abstract Meaning Representation (AMR) (Banarescu et al., 2013) parsing, a semantic framework representing sentence meaning as directed graphs. We test models on 100 randomly selected samples from the LDC dataset (Bonn et al.), which contains 39,260 sentence-AMR pairs. We compare models with and without the GRAMMAR-LLM module in terms of SMATCH F1 score (Cai and Knight, 2013), which measures structural similarity between AMR graphs, and report the percentage of valid graphs

Model	Smatch	Validity
AMRBART-base	0.559	94%
G-AMRBART-base	**0.577**	100%
AMRBART-large	0.789	100%
G-AMRBART-large	**0.790**	100%

Table 4: Performance of the models on the AMR semantic parsing task, evaluated on the LDC dataset in terms of Smatch metric.

generated after fine-tuning and postprocessing, as AMR-BART useS a rule-based approach to rectify invalid outputs. As shown in Table 4, our G-AMRBART-base outperforms its unconstrained counterpart (AMRBART-base) by 2 percentage points. Meanwhile, G-AMRBART-large exhibits comparable performance to AMRBART-large, achieving a SMATCH F1 score of 0.789 compared to 0.79. The improved performance of the larger model can be attributed to its increased parameter count and extensive training data, enabling more effective fine-tuning and enhanced ability to process and learn the syntactic structure of AMR graphs. In this scenario, our method shows comparable performance with the unconstrained fine-tuned model, as no significant further enhancements are attainable. In contrast, the AMRBART-base model, constrained by its reduced parameter count, struggles to learn the syntactic rules, leading to invalid graph outputs in 6% of cases. Here, the integration of a grammar module ensures the generation of semantically valid outputs, thereby outperforming the baseline model.

5 Conclusion

We present GRAMMAR-LLM, a novel framework that integrates formal grammatical constraints into the decoding of language models using LL (prefix) grammars. Our approach dynamically enforces syntactic constraints during generation in real-time, with minimal computational cost. We tested our approach in hierarchical classification, sign language translation, and semantic parsing, demonstrating significant improvements in classification accuracy, translation quality, and adherence to structured output constraints.

6 Limitations

Our approach inherits the autoregressive nature of LLMs, where each token generated affects the probability distribution of subsequent outputs. This sequential dependency can lead to error propagation, as early-stage mistakes may significantly impact the quality of later tokens. Our approach could, in certain cases, amplify early-stage mistakes since validity constraints might intervene later on, preventing the generation of tokens which are necessary to complete a meaningful (though invalid) sentence. Techniques such as those proposed in Park et al. (2024) mitigate early-stage errors by reshaping token distributions. We consider this work complementary to ours and advocate for the synergistic use of both approaches as future work.

Another limit is given by the expressiveness of our class of LL(prefix) grammars. Although more expressive than traditional LL(1) grammars, LL(prefix) grammars remain less expressive than CFGs. CFGs allow for grammar ambiguity and can model more complex syntactic structures, which may be necessary for certain advanced natural language generation tasks. This limitation means that while LL(prefix) grammars are well-suited for many structured generation tasks, they may struggle with highly recursive or nested language constructs that require the full power of CFGs. The downside of using the entire class of CFGs is that their languages cannot be accepted left-to-right by a deterministic PDA. Consequently, integrating them with LLMs (Geng et al., 2023) would introduce complex data structures and a superlinear overhead, with the exponent depending on the specific grammar (Angelov, 2009).

Furthermore, our current implementation does not incorporate an end-to-end fine-tuning phase that integrates the grammar module into the training loop. As a result, our findings may not fully capture the potential behavior of LLMs when fine-tuned on extensive datasets, particularly in scenarios where the grammar constraints are deeply embedded in the model's learning process. Exploring grammar-constrained fine-tuning remains an avenue for future research.

Acknowledgement

We acknowledge financial support from the Next Generation EU Program with the Future Artificial Intelligence Research (FAIR) project, code PE00000013, CUP 53C22003630006, and by the PNRR project Learning for All (L4ALL) funded by the Italian MIMIT (number: F/310072/01-05/X56). We thank Lorenzo Anastasi for his valuable support in setting up the infrastructure for our experiments.

References

Kareem Ahmed, Kai-Wei Chang, and Guy Van den Broeck. 2024. Controllable generation via locally constrained resampling. *arXiv preprint arXiv:2410.13111*.

Alfred V Aho, Ravi Sethi, and Jeffrey D. Ullman. 2006. *Compilers: Principles, Techniques, and Tools*. Pearson Education, Inc.

Krasimir Angelov. 2009. Incremental parsing with parallel multiple context-free grammars. In *Proceedings of the 12th conference of the European chapter of the ACL (EACL 2009)*, pages 69–76.

Xuefeng Bai, Yulong Chen, and Yue Zhang. 2022. Graph pre-training for amr parsing and generation. *arXiv preprint arXiv:2203.07836*.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

Julia Bonn, Skatje Myers, Jens Van Gysel, Lukas Denk, Meagan Vigus, Jin Zhao, Andrew Cowell, William Croft, Jan Hajic, Martha Palmer, et al. Abstract meaning representation (amr) annotation release 3.0.

Shu Cai and Kevin Knight. 2013. Smatch: an evaluation metric for semantic feature structures. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 748–752.

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.

Yixin Dong, Charlie F Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. 2024. Xgrammar: Flexible and efficient structured generation engine for large language models. *arXiv preprint arXiv:2411.15100*.

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

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert

West, Eric Horvitz, and Harsha Nori. 2025. Generating structured outputs from language models: Benchmark and studies. *arXiv preprint arXiv:2501.10868*.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-constrained decoding for structured nlp tasks without finetuning. *arXiv preprint arXiv:2305.13971*.

John E Hopcroft, Rajeew Motwani, and Jeffrey D Ullman. 2001. *Introduction to automata theory, languages and computation*. Addison-Wesley.

Aishwarya Kamath and Rajarshi Das. 2018. A survey on semantic parsing. *arXiv preprint arXiv:1812.00978*.

Terry Koo, Frederick Liu, and Luheng He. 2024. Automata-based constraints for language model decoding. *arXiv preprint arXiv:2407.08103*.

Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. 2017. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE.

Xianming Li and Jing Li. 2023. Angle-optimized text embeddings. *arXiv preprint arXiv:2309.12871*.

Zelong Li, Wenyue Hua, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Formal-llm: Integrating formal language and natural language for controllable llm-based agents. *arXiv preprint arXiv:2402.00798*.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

Adrián Núñez-Marcos, Olatz Perez-de Viñaspre, and Gorka Labaka. 2023. A survey on sign language machine translation. *Expert Systems with Applications*, 213:118993.

Achraf Othman and Mohamed Jemni. 2012. English-asl gloss parallel corpus 2012: Aslg-pc12. In *5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon LREC*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D'Antoni. 2024. Grammar-aligned decoding. *arXiv preprint arXiv:2405.21047*.

Kanghee Park, Timothy Zhou, and Loris D'antoni. 2025. Flexible and efficient grammar-constrained decoding.

Maja Popović. 2015. chrf: character n-gram f-score for automatic mt evaluation. In *Proceedings of the tenth workshop on statistical machine translation*, pages 392–395.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*.

Richard Shin, Christopher H 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. *arXiv preprint arXiv:2104.08768*.

Carlos N Silla and Alex A Freitas. 2011. A survey of hierarchical classification across different application domains. *Data mining and knowledge discovery*, 22:31–72.

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

Honghua Zhang, Meihua Dang, Nanyun Peng, and Guy Van den Broeck. 2023. Tractable control for autoregressive language generation. In *International Conference on Machine Learning*, pages 40932–40945. PMLR.

Honghua Zhang, Po-Nien Kung, Masahiro Yoshida, Guy Van den Broeck, and Nanyun Peng. 2024. Adaptable logical control for large language models. *arXiv preprint arXiv:2406.13892*.