# Grammar-Constrained Decoding Makes Large Language Models Better Logical Parsers

**Federico Raspanti** **Tanir Ozcelebi** **Mike Holenderski**
Eindhoven University of Technology, Eindhoven, The Netherlands
{f.raspanti, t.ozcelebi, m.holenderski}@tue.nl

## Abstract

Large Language Models (LLMs) have shown capabilities in various natural language processing tasks, yet they often struggle with logical reasoning, particularly when dealing with complex natural language statements. To address this challenge, approaches that combine LLMs with symbolic reasoners have been proposed, where the LLM translates the natural language statements into symbolic representations, which are then verified by an external symbolic solver. However, ensuring syntactic correctness in these translations remains a significant challenge. To address this, we propose to constrain the outputs of the LLMs using Grammar Constrained Decoding (GCD), showing that it consistently improves both syntactic correctness and accuracy in logical parsing tasks. Our findings demonstrate that grammar constraints can complement in-context examples, especially beneficial for resource-constrained applications using smaller models. However, we observe that while GCD ensures syntactic validity, semantic errors not captured by Context-Free Grammars continue to pose challenges. Additionally, our results reveal a trade-off for larger models where unconstrained generation occasionally outperforms constrained decoding, aligning with recent theoretical work on bias introduced by constrained decoding. Our code and data is publicly available at: https://github.com/federaspa/gcd-llm-logical-parsing

## 1 Introduction

In recent years, Large Language Models (LLMs) (Devlin et al., 2019; Brown et al., 2020; Achiam et al., 2023; Team et al., 2023; The; Touvron et al., 2023) have shown increasing capabilities for logical reasoning, especially when guided with prompting techniques such as few-shot examples (Parnami and Lee, 2022) and Chain-of-Thought (CoT) prompting (Wei et al., 2022).

The reasoning capabilities of these models have traditionally been evaluated on standardized benchmarks like GSM8K (Cobbe et al., 2021), where LLMs are tasked with solving an arithmetic problem, demonstrating increasingly impressive performance. This apparent progress has led to optimistic interpretations about LLMs' ability to perform genuine reasoning.

However, recent studies have shown that polluting problems, by randomly selecting symbols or adding irrelevant information, significantly degrades performance in all state-of-the-art models (Mirzadeh et al., 2024). These findings indicate that rather than developing true reasoning capabilities, LLMs may primarily be learning to reproduce training examples with minor variations.

To tackle this challenge, an increasingly popular approach is to *decouple* the reasoning process, using LLM to convert natural language problems into symbolic representations, treating them as logical *parsers*, and then using symbolic solvers to determine the outcome of the logical problem (e.g. True, False, and in some cases Undecidable) (Pan et al., 2023; Feng et al., 2023; Wang et al., 2024).

This approach has been shown to increase accuracy on symbolic reasoning tasks, but introduces the new challenge of respecting the syntax required by the solver when converting the problems into symbolic representations, which has typically been addressed in two, non-mutually exclusive ways: by providing in-context examples to the LLM (In-Context Learning, ICL), and by relying on the LLM's ability to identify and correct its own mistakes (Self-Verification) (Pan et al., 2023; Wang et al., 2024; Feng et al., 2023). Both solutions were proven to be effective in improving syntactic correctness, but neither provides strong guarantees.

In this context, GCD emerges as a promising approach to *guarantee* syntactic correctness in symbolic representations. GCD works by dynamically constraining the model's output space during generation, ensuring that only grammatically valid sequences can be produced (Geng et al., 2024b; Park et al., 2024). This approach differs from previous methods in that it provides deterministic guarantees about the syntax of the generated output.

Recent findings (Tam et al., 2024) demonstrated that grammar constraints can significantly degrade LLM reasoning abilities when reasoning is performed directly by the language model. This raises the question of whether this still holds when decoupling the reasoning process. We hypothesize that, when using LLMs strictly as parsers and delegating the reasoning to specialized solvers, the constraints on generation will increase the syntactic correctness of the symbolic representations, which will in turn increase downstream accuracy.

This paper focuses on the following research questions (RQs).

**RQ1.** *Can GCD improve the performance of LLMs as logical parsers, measured by accuracy on a downstream task?*

**RQ2.** *How effective is GCD for compensating in-context learning, measured by accuracy on a downstream task?*

**RQ3.** *How does the impact of GCD vary with model size, measured by accuracy on a downstream task?*

The paper is organized as follows. Section 2 discusses related work in LLMs as logical solvers and GCD. Section 3 introduces our methodology. Section 4 introduces our experimental setup and evaluation methodology. Section 5 presents our main results and empirical findings, discussed in Section 6. Section 7 presents a summary of our contributions and findings. Finally, Section 8 concludes with the limitations of our approach and discusses future work.

## 2 Related Work

### 2.1 Logical Reasoning with LLMs

The development of logical reasoning capabilities in LLMs has seen significant progress through various approaches. Wei et al. (2022) introduced Chain-of-Thought (CoT) prompting to break down complex reasoning into steps, while Kojima et al. (2023) demonstrated that simply prompting LLMs to "think step by step" could achieve similar results without examples. To address inconsistencies in LLMs' logical reasoning, Creswell et al. (2022) developed the Faithful Reasoning framework, combining LLMs with automated reasoning tools.

Recent research has focused on integrating LLMs with symbolic solvers, treating LLMs as logical *parsers* rather than *reasoners*. Pan et al. (2023) introduced Logic-LM, which combines LLMs (GPT-3.5-Turbo, GPT-4-Turbo) with symbolic solvers (Prover9, Z3, Pyke) and includes a self-refinement loop to handle invalid formulas. Wang et al. (2024) developed ChatLogic, integrating LLMs with a pyDatalog reasoning engine and incorporating semantic and syntax correction modules. While their approach attempts to guide syntax corrections through prompting, they noted that these corrections were unreliable. We propose to address this limitation by enforcing syntax using GCD, with the aim of improving the reliability of problem generation.

### 2.2 Grammar-Constrained Decoding

GCD has emerged as an effective method for constraining LLM outputs to respect user-defined rules, particularly when models haven't been extensively trained on domain-specific syntax. Two main approaches have been developed to achieve grammatical adherence: grammar prompting, which guides LLMs to follow specific grammars like those written in Backus-Naur form, and GCD itself, which directly constrains the decoding process (Wang et al., 2023).

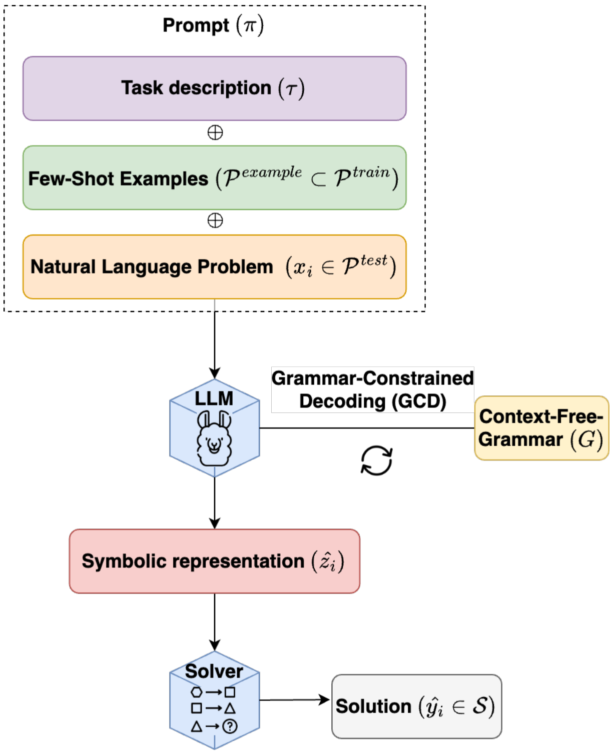
At the core of GCD is a Context-Free Grammar (CFG), which consists of non-terminals ($V$), terminals ($E$), production rules ($R$), and a starting symbol ($S$). A simple example of such a grammar is shown below:

```
S ::= NP VP
NP ::= Det N
VP ::= V NP
Det ::= "the" | "a"
N ::= "cat" | "dog"
V ::= "chases" | "sees"
```

Listing 1: An example of a CFG grammar

During the decoding process, the language model's output is restricted to sequences that can be derived from the defined grammar. The model's vocabulary is filtered to include only grammatically valid tokens at each step, with probabilities redistributed among these options. This process involves expanding non-terminals and backtracking when necessary until a complete, syntactically correct sequence is generated.

Early work in this field includes GrammarCNN (Sun et al., 2019), which incorporated grammar



Prompt $(\pi)$
Task description $(\tau)$
$\oplus$
Few-Shot Examples $(\mathcal{P}^{example} \subset \mathcal{P}^{train})$
$\oplus$
Natural Language Problem $(x_i \in \mathcal{P}^{test})$
LLM
Grammar-Constrained Decoding (GCD)
Context-Free-Grammar $(G)$
Symbolic representation $(\hat{z}_i)$
Solver
Solution $(\hat{y}_i \in \mathcal{S})$

Figure 1: Full pipeline. The LLM processes a prompt made of a task description $\tau$, examples $\mathcal{P}^{\text{example}}$, and a Natural Language problem $x_i$. We use GCD to generate syntactically valid symbolic representations $\hat{z}_i$, which are evaluated by a solver to produce the final solution $\hat{y}_i$.

knowledge into convolutional neural networks, and the CTRL model (Keskar et al., 2019), which used control codes to generate text with specific attributes. However, CTRL's approach was limited by the need to train the model on selected control codes, making it less suitable for domain-specific or data-scarce fields.

More recent developments include the sketch-based method of Geng et al. (2024a), where a grammar-constrained LLM rewrites the output of a powerful black-box model.

Recently Tam et al. (2024) demonstrated that grammar constraints can significantly degrade LLM reasoning abilities. However, while they argue for avoiding such constraints to preserve reasoning capabilities of LLMs, we argue that the benefits of reliable structured output outweigh the potential reasoning degradation if we focus LLM on parsing and delegate the reasoning to a symbolic solver.

## 3 Method

We illustrate our methodology in Figure 1.

**Step 1: Problem formulation**

In the first step, we use an LLM to extract symbolic representations of natural language problems.

Consider two labeled sets of problems, $\mathcal{P}^{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N^{\text{train}}}$ and $\mathcal{P}^{\text{test}} = \{(x_i, y_i)\}_{i=1}^{N^{\text{test}}}$, where $x_i$ is a problem expressed in natural language, and $y_i \in \mathcal{S}$ is the ground truth solution to this problem in some domain $\mathcal{S}$. Let $z_i$ be a symbolic representation of the problem $x_i$ (there may be many valid representations for a given $x_i$).

Let $G$ be a Context-Free Grammar, $\tau$ a task description, and $\mathcal{P}^{\text{example}}$ a set of examples, with $\mathcal{P}^{\text{example}} \subset \mathcal{P}^{\text{train}}$.

Let $\pi_i = \tau \oplus \mathcal{P}^{\text{example}} \oplus x_i$ be the prompt for the problem $x_i$, where $\oplus$ denotes concatenation [Section A]. Then, we define $\hat{z}_i$ as:

$$\hat{z}_i = \text{LLM}(\pi_i; G) \tag{1}$$

where $\text{LLM}(\cdot; G)$ is a function that takes a prompt as input and produces output that is consistent with grammar $G$.

**Step 2: Problem solution**

Let $\text{Solver}(\hat{z}_i) \in \mathcal{S} \cup \{\perp\}$ be the solution returned by the symbolic solver to a problem in its symbolic representation $\hat{z}_i$, where $\text{Solver}(\hat{z}_i) = \perp$ indicates that $\hat{z}_i$ is invalid, i.e., it contained a syntax error and could not be solved. We define the predicted solution $\hat{y}_i$ as:

$$\hat{y}_i = \text{Solver}(\hat{z}_i) \tag{2}$$

## 4 Experiments and evaluation

### 4.1 Experiments

We designed three experiments to evaluate three different aspects of GCD for LLMs as logical parsers. First, we compare outputs between unconstrained generation (Unc.) and generation constrained by domain-specific grammar (Const.). Second, we investigate how well GCD can compensate for In-Context Learning, by combining grammar constraints with zero-shot, two-shot, and five-shot prompting. Third, we assess the impact of GCD and In-Context Learning across models of varying parameter counts. We measure performance in terms of semantic accuracy (comparing solver outputs to ground truth) and syntactic accuracy (percentage of generated programs that parse without errors), as described in Sec. 4.3. For each experiment, we perform independent runs and report the mean and standard deviation of the results in Tables 1 and 2.

## 4.2 Models

We selected open-source LLMs from four families: Gemma (2B, 9B, 27B), Llama (1B, 3B, 8B), Mistral (8B, 22B), and Qwen (0.5B, 1.5B, 3B, 7B, 14B). Within each family, we chose variants of different parameter counts, to investigate GCD's impact across a different model architectures and sizes. All models are instruction-tuned variants.

## 4.3 Metrics

We measure the semantic accuracy (**Accuracy**, Eq. 3) of the predicted symbolic representation by running all programs through the symbolic solver and comparing the result with the ground truth. We consider failure to parse the symbolic representation (i.e. the solver returning an error) as a wrong answer. We also measure the syntactic accuracy (**Executable Rate**, Eq. 4) of generated programs by observing the fraction of generated programs that the solver can run without incurring an error.

$$\text{Accuracy} = \frac{\sum_{i=1}^{N} \mathbf{1}(\hat{y}_i = y_i)}{N} \tag{3}$$

$$\text{Executable Rate} = \frac{\sum_{i=1}^{N} \mathbf{1}(\hat{y}_i \neq \perp)}{N} \tag{4}$$

where $\mathbf{1}$ is the indicator function (1 if true, 0 if false).

First we highlight that, since there can only be as many correct answers as valid symbolic representations, Accuracy $\leq$ Executable Rate.

Second, we highlight that **we may achieve 0 Accuracy if none of the symbolic representations were valid**. This does not mean that flipping all predictions would yield perfect accuracy, but rather indicates complete failure at producing syntactically valid formulas that the solver can process.

Finally, we note that while GCD ensures that generated outputs conform to the specified grammar, semantic errors can still occur that prevent successful execution. These semantic errors are not captured by the CFG but still result in solver failures ($\hat{y}_i = \perp$). For instance, in FOL generation, a predicate with the same name may appear with different arities in the same problem (e.g., `Predicate(x)` and `Predicate(x, y)`) or in arithmetic problems, variable references might refer to variables not previously declared in the problem. This explains why even with grammar constraints, we observe executable rates below 1.0, particularly for smaller models that may struggle with maintaining semantic consistency.

## 4.4 Datasets and Solvers

We evaluate the proposed method on two datasets that contain problems from two branches of mathematics: first-order logic (FOL) and arithmetic.

### First-order logic

For FOL, we chose *FOLIO* (Han et al., 2024), a dataset for logical reasoning constructed by domain experts. The problems incorporate real-world knowledge with natural language formulations, requiring complex logic reasoning to get a solution. Our evaluation utilizes the complete FOLIO test set, comprising 204 distinct examples.

For the solver, we chose *Prover9* (McCune, 2005–2010), a widely accepted automated theorem prover for FOL. Following the implementation approach of Pan et al. in Logic-LM (Pan et al., 2023), we integrated Prover9 into our pipeline through Python's NLTK library, to evaluate both the syntactic correctness and the outcome of the generated formulas.

### Arithmetic

For arithmetic, we chose *GSM-symbolic* (Mirzadeh et al., 2024), a dataset derived from the GSM8K (Cobbe et al., 2021) math word problem benchmark, where the problems are reformulated to account for data contamination in previously released LLMs. The problems incorporate arithmetical knowledge with natural language formulations. This evaluation utilizes a subset of 1000 randomly sampled samples from the GSM-symbolic test set.

For the solver, we used SymPy, a Python library for symbolic arithmetic. We generate the problems in standard infix notation (SIN) and implement a wrapper around SymPy to parse and evaluate the symbolic representations.

## 5 Results

We report the average results of our runs in Tables 1 and 2, showing the impact of GCD on accuracy and executable rate respectively. Our results indicate that grammatical constraints provide the most benefits to smaller models and in resource-constrained scenarios where few or no examples are available.

| | FOLIO | | | | | | GSM-symbolic | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0-shots | | 2-shots | | 5-shots | | 0-shots | | 2-shots | | 5-shots | |
| **Model** | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* |
| gemma2-2b | 0.02 | **0.21** | 0.07 | **0.19** | 0.06 | **0.24** | 0.00 | **0.15** | 0.18 | **0.20** | 0.18 | **0.21** |
| gemma2-9b | 0.23 | **0.51** | 0.46 | **0.51** | 0.50 | **0.51** | 0.17 | **0.25** | **0.44** | 0.39 | **0.41** | 0.37 |
| gemma2-27b | 0.40 | **0.50** | 0.49 | **0.56** | 0.51 | **0.55** | **0.31** | 0.30 | **0.54** | 0.49 | **0.51** | 0.49 |
| llama3.2-1b | 0.00 | **0.19** | 0.00 | **0.15** | 0.01 | **0.20** | 0.00 | **0.03** | 0.01 | **0.02** | 0.01 | **0.03** |
| llama3.2-3b | 0.00 | **0.27** | 0.08 | **0.23** | 0.12 | **0.25** | 0.00 | **0.12** | 0.13 | **0.18** | 0.16 | **0.19** |
| llama3.1-8b | 0.05 | **0.28** | 0.19 | **0.33** | 0.27 | **0.36** | 0.00 | **0.27** | 0.30 | **0.37** | 0.28 | **0.35** |
| ministral-8b | 0.05 | **0.29** | 0.12 | **0.27** | 0.15 | **0.28** | 0.01 | **0.12** | 0.26 | **0.27** | 0.26 | **0.28** |
| mistral-22b | 0.22 | **0.41** | 0.41 | **0.45** | 0.40 | **0.47** | 0.00 | **0.13** | **0.42** | 0.38 | **0.42** | 0.39 |
| qwen2.5-0.5b | 0.00 | **0.14** | 0.02 | **0.20** | 0.05 | **0.22** | 0.00 | **0.01** | 0.01 | 0.01 | 0.02 | 0.02 |
| qwen2.5-1.5b | 0.00 | **0.20** | 0.05 | **0.22** | 0.08 | **0.23** | 0.00 | **0.05** | 0.06 | **0.08** | 0.07 | **0.09** |
| qwen2.5-3b | 0.01 | **0.29** | 0.16 | **0.22** | 0.19 | **0.28** | 0.00 | **0.09** | 0.18 | **0.33** | 0.17 | **0.31** |
| qwen2.5-7b | 0.21 | **0.33** | 0.31 | **0.32** | **0.39** | 0.35 | 0.00 | **0.20** | 0.44 | **0.45** | 0.46 | **0.47** |
| qwen2.5-14b | 0.18 | **0.29** | 0.33 | **0.31** | **0.36** | 0.26 | 0.29 | **0.37** | **0.57** | 0.38 | **0.56** | 0.36 |

Table 1: Accuracy of LLMs as logical parsers across different model sizes and prompting strategies (0-shot, 2-shot, 5-shot) with unconstrained (Unc.) versus grammar-constrained (Con.) decoding on GSM-symbolic and FOLIO datasets. As highlighted in Section 4.3, Accuracy $\leq$ Executable Rate in Table 2. We may achieve zero Accuracy if all the symbolic representations were invalid.

### 5.1 Grammar Constraints

Both in terms of accuracy and executable rate FOL syntax constraints outperform the unconstrained baseline. The impact is most significant when looking at executable rate, where FOL constraints achieve above 0.70 executable rate even with smaller models that show near-zero executable rate in unconstrained conditions. For small models like gemma2-2b and qwen2.5-3b, after introducing the constraints, we go from producing almost no executable outputs to achieving high rates of executable outputs.

### 5.2 In-Context Learning

Few-shot prompting enhances accuracy and executable rate across all settings. We observe that in many cases the relative improvement from introducing few-shot examples is smaller with GCD compared to the unconstrained baseline. Moreover, we observe that, in most cases, GCD with 0 shots achieves higher accuracy than unconstrained decoding with 5 shots, and comparable accuracy to GCD and 5 shots. This indicates that GCD can compensate for the absence of examples in the 0 shots setting.

### 5.3 Model Size

The benefits of GCD can be observed on all model sizes, although they are proportionally more significant for smaller models and fewer shots. For instance, with 0 shots, smaller models show more improvements in accuracy when using GCD compared to their larger counterparts. The largest models in our test suite show increases in accuracy with zero-shot prompting, but show diminishing returns when example shots are increased.

Notably, for the largest models ($\geq$ 14B) with multiple shots, we observe instances where unconstrained decoding achieves comparable or occasionally greater accuracy (Table 1), suggesting that model capacity and number of examples can influence the effectiveness of grammar constraints.

This pattern becomes clearer when comparing accuracy with executable rates: while larger models maintain high executable rates under constraints, their accuracy sometimes decreases, suggesting a trade-off between syntactic validity and semantic correctness.

## 6 Discussion

### RQ1.

Our results show that GCD improves the performance of LLMs, when they are used as logical parsers. The experiments show consistent improvements in both accuracy and executable rate across model sizes and number of examples.

This improvement in parsing execution rate directly translates to improved reasoning of the overall system, since the symbolic solver can only

| | FOLIO | | | | | | GSM-symbolic | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0shots | | 2shots | | 5shots | | 0shots | | 2shots | | 5shots | |
| **Model** | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* |
| gemma2-2b | 0.07 | **0.60** | 0.19 | **0.52** | 0.16 | **0.65** | 0.00 | **1.00** | 0.78 | **1.00** | 0.76 | **1.00** |
| gemma2-9b | 0.41 | **0.90** | 0.64 | **0.84** | 0.73 | **0.83** | 0.43 | **1.00** | 0.93 | **0.99** | 0.93 | **0.99** |
| gemma2-27b | 0.67 | **0.94** | 0.74 | **0.92** | 0.79 | **0.89** | 0.64 | **0.99** | 0.96 | **1.00** | 0.96 | **1.00** |
| llama3.2-1b | 0.00 | **0.57** | 0.00 | **0.43** | 0.01 | **0.62** | 0.00 | **0.98** | 0.27 | **0.98** | 0.24 | **0.98** |
| llama3.2-3b | 0.00 | **0.72** | 0.19 | **0.59** | 0.25 | **0.64** | 0.00 | **0.99** | 0.70 | **1.00** | 0.76 | **1.00** |
| llama3.1-8b | 0.09 | **0.78** | 0.38 | **0.77** | 0.43 | **0.78** | 0.00 | **0.99** | 0.76 | **1.00** | 0.76 | **1.00** |
| ministral-8b | 0.09 | **0.83** | 0.32 | **0.76** | 0.37 | **0.77** | 0.02 | **0.99** | 0.83 | **1.00** | 0.83 | **1.00** |
| mistral-22b | 0.40 | **0.87** | 0.72 | **0.86** | 0.69 | **0.86** | 0.00 | **0.99** | 0.93 | **1.00** | 0.93 | **1.00** |
| qwen2.5-0.5b | 0.00 | **0.40** | 0.07 | **0.58** | 0.13 | **0.65** | 0.00 | **0.94** | 0.58 | **0.98** | 0.53 | **0.98** |
| qwen2.5-1.5b | 0.01 | **0.56** | 0.14 | **0.58** | 0.18 | **0.58** | 0.01 | **0.97** | 0.65 | **0.99** | 0.69 | **0.97** |
| qwen2.5-3b | 0.04 | **0.75** | 0.29 | **0.54** | 0.37 | **0.65** | 0.00 | **0.97** | 0.45 | **0.98** | 0.46 | **0.98** |
| qwen2.5-7b | 0.37 | **0.72** | 0.60 | **0.67** | 0.64 | **0.73** | 0.01 | **0.96** | 0.83 | **1.00** | 0.87 | **1.00** |
| qwen2.5-14b | 0.30 | **0.71** | 0.62 | **0.72** | **0.65** | 0.62 | 0.59 | **1.00** | 0.95 | **0.99** | 0.96 | **0.99** |

Table 2: Executable Rate of LLMs as logical parsers across different model sizes and prompting strategies (0-shot, 2-shot, 5-shot) with unconstrained (Unc.) versus grammar-constrained (Con.) decoding on GSM-symbolic and FOLIO datasets.

process syntactically valid formulas. This enables more problems to be successfully processed through the complete reasoning pipeline, resulting in higher end-to-end accuracy on logical reasoning tasks.

### RQ2.

We show that models using GCD with zero-shot prompting achieve only slightly lower performance compared to unconstrained models using five-shot prompting. This can be valuable in domains where creating high-quality examples requires expert knowledge or where prompt length limitations do not allow for demonstrations.

However, our findings also indicate that GCD and in-context learning are complementary rather than mutually exclusive approaches. The highest performance was often achieved by combining GCD with multiple examples, indicating that, while GCD can compensate for limited examples, it does not fully replicate the guidance provided by in-context learning. This suggests that, when resources permit, practitioners should consider implementing both strategies.

### RQ3.

Smaller models experience greater improvements from GCD compared to their larger counterparts. This finding indicates that GCD could help democratize logical parsing capabilities by making smaller, more accessible models perform more reliably.

However, our findings also reveal that for larger models with few-shot examples, unconstrained generation occasionally outperforms constrained decoding. This phenomenon has been theoretically and empirically validated by recent work. Ye et al. (2025) proved that constrained decoding introduces bias into output distributions, demonstrating a significant KL-divergence between the true distribution and the constrained decoding distribution. We hypothesize that, for smaller models, grammatical constraints can skew the distribution of the outputs towards more appropriate ones, but as models grow in size their learned representations become "good enough" to perform the parsing, and the bias introduced by the constraints degrades the output.

## 7 Conclusion

In this work, we investigated the effectiveness of GCD for improving Large Language Models when used as logical parsers in problem-solving pipelines. By separating the parsing task from the reasoning process and delegating logical inference to symbolic solvers, we examined whether syntactic constraints could improve the accuracy of these systems.

Our experiments across thirteen open-source LLMs, ranging from 0.5B to 27B parameters, demonstrate that GCD significantly improves syntactic correctness and downstream semantic accuracy. We found that smaller models benefit most from grammatical constraints, with models like gemma2-2b achieving executable rates above 60% in FOL tasks when constrained, compared to near-zero rates without constraints. This pattern suggests that GCD could democratize logical parsing capabilities by enabling smaller, more resource-efficient models to perform reliably in formal reasoning tasks.

The results also reveal that GCD can effectively compensate for limited in-context examples. In many cases, zero-shot prompting with grammar constraints achieved comparable or superior performance to five-shot unconstrained generation. This finding has practical implications for domains where expert-annotated examples are scarce or expensive to obtain. However, we observed that GCD and in-context learning are complementary approaches, with the highest performance often achieved by combining both strategies.

Our work contributes to the broader discussion about the role of syntactic guidance in language model generation. While recent theoretical work suggests that constraints may introduce bias and reduce reasoning capabilities, our empirical results indicate that this trade-off can be beneficial when models are used specifically as parsers rather than reasoners. Using LLMs for natural language understanding and symbolic solvers for logical inference appears to be a promising direction for building more reliable AI systems that can handle formal reasoning tasks.

## 8 Limitations

First, our implementation relies on CFGs that cannot capture context-sensitive constraints found in some reasoning tasks. While GCD based on CFGs improves syntactic correctness, guaranteeing semantic accuracy remains challenging. Our approach significantly increases syntactic validity and downstream semantic accuracy, but it does not ensure that the generated formulas correctly capture the meaning of natural language statements. As noted in Section 5, even with grammar constraints, executable rates below 1.0 indicate the presence of semantic errors that pass syntactic validation but fail during solver execution. For instance, predicate consistency violations, variable scope constraints, and other semantic requirements that extend beyond CFG expressivity continue to pose challenges. Future work could explore extensions to context-sensitive grammars or integration with semantic verification systems.

Second, our evaluation focused on two specific branches of mathematics: FOL and arithmetic reasoning. While these domains demonstrate the approach's effectiveness, extending to other branches of mathematics or fields entirely, such as computational chemistry or physics, would require domain-specific grammar definitions and may reveal additional challenges.

Third, we observed that larger models with few-shot examples occasionally exhibit performance degradation under constraints. As discussed in Section 6, this aligns with theoretical work by Ye et al. (2025) showing that constrained decoding introduces bias into output distributions. This suggests that the benefits of GCD may be model-dependent.

Finally, our approach uses statically defined grammars that remain fixed throughout execution. Adaptive grammars that evolve based on solver feedback or parsing errors could potentially improve performance. Additionally, incorporating semantic information from partial parses to optimize grammar rules based on task performance could address limitations in capturing complex logical relationships (Loula et al., 2025; Albinhassan et al., 2025).

## Acknowledgement

This work was supported by the SmartEM and AIMS5.0 projects. The project SmartEM is supported by the ITEA cluster under grant agreement no. 22009. The project AIMS5.0 is supported by the Chips Joint Undertaking and its members, including the top-up funding by National Funding Authorities from involved countries under grant agreement no. 101112089.

## References

The claude 3 model family: Opus, sonnet, haiku.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

M. Albinhassan, P. Madhyastha, and A. Russo. 2025. Sem-ctrl: Semantically controlled decoding. *arXiv preprint*, arXiv:2503.01804v2.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind

Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2022. Selection-inference: Exploiting large language models for interpretable logical reasoning. *Preprint*, arXiv:2205.09712.

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

Jiazhan Feng, Ruochen Xu, Junheng Hao, Hiteshi Sharma, Yelong Shen, Dongyan Zhao, and Weizhu Chen. 2023. Language models can be logical solvers. *Preprint*, arXiv:2311.06158.

GBNF Guide. Ggerganov/llama.cpp, GBNF Guide. [link].

Saibo Geng, Berkay Döner, Chris Wendler, Martin Josifoski, and Robert West. 2024a. Sketch-guided constrained decoding for boosting blackbox large language models without logit access. *Preprint*, arXiv:2401.09967.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2024b. Grammar-constrained decoding for structured nlp tasks without finetuning. *Preprint*, arXiv:2305.13971.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alex Wardle-Solano, Hannah Szabo, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Alexander R. Fabbri, Wojciech Kryscinski, Semih Yavuz, Ye Liu, Xi Victoria Lin, Shafiq Joty, Yingbo Zhou, Caiming Xiong, Rex Ying, Arman Cohan, and Dragomir Radev. 2024. Folio: Natural language reasoning with first-order logic.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *Preprint*, arXiv:1909.05858.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large language models are zero-shot reasoners. *Preprint*, arXiv:2205.11916.

J. Loula, B. LeBrun, L. Du, B. Lipkin, C. Pasti, G. Grand, T. Liu, Y. Emara, M. Freedman, J. Eisner, R. Cotterell, V. Mansinghka, A. K. Lew, T. Vieira, and T. J. O'Donnell. 2025. Syntactic and semantic control of large language models via sequential monte carlo. In *International Conference on Learning Representations (ICLR)*.

W. McCune. 2005–2010. Prover9 and mace4. `http://www.cs.unm.edu/~mccune/prover9/`.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *Preprint*, arXiv:2305.12295.

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

Archit Parnami and Minwoo Lee. 2022. Learning from few examples: A summary of approaches to few-shot learning. *arXiv preprint arXiv:2203.04291*.

Recursive Grammar Issue. Ggerganov/llama.cpp, Issue #7572, "Bug: GBNF repetition rewrite results in unsupported left recursion". [link].

Zeyu Sun, Qihao Zhu, Lili Mou, Yingfei Xiong, Ge Li, and Lu Zhang. 2019. A grammar-based structural cnn decoder for code generation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7055–7062.

Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung yi Lee, and Yun-Nung Chen. 2024. Let me speak freely? a study on the impact of format restrictions on performance of large language models. *Preprint*, arXiv:2408.02442.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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. *Preprint*, arXiv:2305.19234.

Zhongsheng Wang, Jiamou Liu, Qiming Bao, Hongfei Rong, and Jingfeng Zhang. 2024. Chatlogic: Integrating logic programming with large language models for multi-step reasoning. *Preprint*, arXiv:2407.10162.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

H. Ye, H. Jain, C. You, A. T. Suresh, H. Lin, J. Zou, and F. Yu. 2025. Efficient and asymptotically unbiased constrained decoding for large language models. In *Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025*, volume 258. PMLR.

## A Prompts

When designing our prompts, we follow the implementation of (Pan et al., 2023), adapting it to our use-cases. We ask the model to generate its output in JSON format, to facilitate parsing its answers to interact with the symbolic solver by making symbolic rules and questions easy to identify.

When we provide examples in the prompt, we do so in JSON format, to guide the generation in our desired format [Listings 2, 3, 4, 5].

### A.1 First-order-logic prompts

```
### TASK DESCRIPTION ###

The task is to convert a natural
     language reasoning problem into
     first-order logic.
First, identify the predicates and
     constants required to build the
     first order logic formulas.
Then, use them to build the rules and
     the conclusion.
Do not attempt to prove or disprove the
     conclusion, limit yourself to
     converting.

You reply strictly in JSON format, with
     the following schema:
"""
{
"fol_preds": [list of required FOL
     Predicates],
"fol_consts": [list of required FOL
     Constants],
"fol_rules": [list of generated FOL
     Rules],
"fol_conc": [generated FOL Conclusion]
}
"""

### NATURAL LANGUAGE PROBLEM ###

Now let's convert this problem to first-
     order logic:

NL premises:
"""
[[nl_problem]]
"""

NL conclusion:
"""
[[nl_conclusion]]
"""
```

Listing 2: Zero-shot prompt template for generating FOL problems

```
### TASK DESCRIPTION ###

The task is to convert a natural
     language reasoning problem into
     first-order logic.
First, identify the predicates and
     constants required to build the
     first order logic formulas.
Then, use them to build the rules and
     the conclusion.
Do not attempt to prove or disprove the
     conclusion, limit yourself to
     converting.

You reply strictly in JSON format, with
     the following schema:
"""
{
"fol_preds": [list of required FOL
     Predicates],
"fol_consts": [list of required FOL
     Constants],
"fol_rules": [list of generated FOL
     Rules],
"fol_conc": [generated FOL Conclusion]
}
"""

### EXAMPLES ###

Here's an example of how to perform the
     conversion:

[[example1]]

###

Here's another example:

[[example2]]

###

...

### NATURAL LANGUAGE PROBLEM ###

Now let's convert this problem to first-
     order logic:

NL premises:
"""
[[nl_problem]]
"""

NL conclusion:
"""
[[nl_conclusion]]
"""
```

Listing 3: Few-shot prompt template for generating FOL problems

## A.2 Arithmetic prompts

```
### TASK DESCRIPTION ###

The task is to convert a natural
     language reasoning problem into
     standard infix notation.
First, identify all the relevant
     variables and their values or
     expressions.
Then, write each variable assignment in
     standard infix notation.
Finally, formulate the equation to solve
      using these variables, also in
     standard infix notation.
Do not attempt to solve the problem,
     limit yourself to converting


You reply strictly in JSON format, with
     the following schema:
"""
\{
"data": [list of relevant variable
     assignment],
"question": [equation to solve]
\}
"""

### NATURAL LANGUAGE PROBLEM ###

Now let's convert this problem to
     standard infix notation.

""""
[[nl_problem]]
"""
```

Listing 4: Zero-shot prompt template for generating GSM problems

```
### TASK DESCRIPTION ###

The task is to convert a natural
     language reasoning problem into
     standard infix notation.
First, identify all the relevant
     variables and their values or
     expressions.
Then, write each variable assignment in
     standard infix notation.
Finally, formulate the equation to solve
      using these variables, also in
     standard infix notation.
Do not attempt to solve the problem,
     limit yourself to converting


You reply strictly in JSON format, with
     the following schema:
"""
\{
"data": [list of relevant variable
     assignment],
"question": [equation to solve]
\}
"""

### EXAMPLES ###

Here's an example of how to perform the
     conversion:

[[example1]]

###

Here's another example:

[[example2]]

###

...

### NATURAL LANGUAGE PROBLEM ###

Now let's convert this problem to
     standard infix notation.

""""
[[nl_problem]]
"""
```

Listing 5: Few-shot prompt template for generating GSM problems

## B Grammars

We write our grammars in the GBNF (Graydon's BNF) format, a variation of the Backus-Naur Form specifically designed for use with language models (GBNF Guide).

Due to limitations in the llama.cpp library (Recursive Grammar Issue), we modified our approach by unrolling the grammars to handle formulas nested up to arbitrary depth [Listings 6 and 7].

```
#### Wrap data and question in a valid
     JSON ####
root ::= "{" ws data ws quest ws "}"

ws ::= | " " | "\n" [ \t]{0,5}

data ::= "\"data\":" ws "[" ws datalist
     ws "], "
datalist ::= "\"" ASSIGNMENT "\"" (ws
     "," ws "\"" ASSIGNMENT "\"")*

quest ::= "\"question\":" ws "\""
     EXPRESSION "\""


#### Mathematical Expressions ####
ASSIGNMENT ::= variable " = " EXPRESSION

EXPRESSION ::= TERM TAIL{0,5}
TAIL ::= OPERATOR TERM

# Terms can be numbers, variables, or
     parenthesized expressions
TERM ::= number | variable | "("
     EXPRESSION ")"

# Operators
OPERATOR ::= " + " | " - " | " * " | " /
      "

# Basic elements
number ::= [0-9]+ ("." [0-9]+)?
variable ::= [a-z_][a-z0-9_]*
```

Listing 6: Grammar for generating valid SIN problems

```
#### Wrap predicates, constants, rules
     and conclusion in a valid JSON ####
root ::= "{" ws preds ws consts ws rules
      ws conc ws "}"

ws ::= | " " | "\n" [ \t]{0,5}

preds ::= "\"fol_preds\":" ws "[" ws
     predslist ws "], "
predslist ::= "\"" ATOMIC "\"" (ws ","
     ws "\"" ATOMIC "\"")*

consts ::= "\"fol_consts\":" ws "[" ws
     constlist ws "], "
constlist ::= "\"" constant "\"" (ws ","
      ws "\"" constant "\"")*

rules ::= "\"fol_rules\":" ws "[" ws
     rulelist ws "], "
rulelist ::= "\"" FORMULA "\"" (ws ","
     ws "\"" FORMULA "\"")*

conc ::= "\"fol_conc\":" ws "\"" FORMULA
      "\""


#### Generate FOL Formulas ####
FORMULA ::= BASIC TAIL{0,5}
TAIL ::= BINOP BASIC

# Basic formula without recursion
BASIC ::= "¬"? ATOMIC | QUANTIFIED | "¬
     "? "(" FORMULA ")"

# Quantified formulas
QUANTIFIED ::= (quantifier variable " ")
     {1,4} "(" FORMULA ")"
quantifier ::= "∀" | "∃"
variable ::= [a-z]

# Binary operators
BINOP ::= " ⊕ " | " ∨ " | " ∧ " | " → "
     | " ↔ "

# Atomic formulas
ATOMIC ::= predicate "(" terms ")"

# Terms in predicates
terms ::= term | term ", " terms

# Individual terms
term ::= constant | variable

# Basic elements
predicate ::= [A-Z][a-zA-Z0-9]+
constant ::= [a-zA-Z0-9]+
```

Listing 7: Grammar for generating valid FOL problems

## C Detailed results

Tables 3-6 provide performance metrics (Accuracy and Executable Rate) for all evaluated models across both datasets (FOLIO and GSM-symbolic) under different prompting conditions (0-shot, 2-shot, and 5-shot) with both unconstrained and grammar-constrained decoding. All results are presented as mean $\pm$ standard deviation.

| | **FOLIO** | | | | | |
|---|---|---|---|---|---|---|
| | **0shots** | | **2shots** | | **5shots** | |
| **Model** | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* |
| gemma2-2b | $0.02^{\pm0.01}$ | $\mathbf{0.21}^{\pm0.04}$ | $0.07^{\pm0.01}$ | $\mathbf{0.19}^{\pm0.09}$ | $0.06^{\pm0.03}$ | $\mathbf{0.24}^{\pm0.02}$ |
| gemma2-9b | $0.23^{\pm0.01}$ | $\mathbf{0.51}^{\pm0.04}$ | $0.46^{\pm0.06}$ | $\mathbf{0.51}^{\pm0.07}$ | $0.50^{\pm0.01}$ | $\mathbf{0.51}^{\pm0.01}$ |
| gemma2-27b | $0.40^{\pm0.01}$ | $\mathbf{0.50}^{\pm0.01}$ | $0.49^{\pm0.03}$ | $\mathbf{0.56}^{\pm0.02}$ | $0.51^{\pm0.04}$ | $\mathbf{0.55}^{\pm0.03}$ |
| llama3.2-1b | $0.00^{\pm0.00}$ | $\mathbf{0.19}^{\pm0.01}$ | $0.00^{\pm0.00}$ | $\mathbf{0.15}^{\pm0.07}$ | $0.01^{\pm0.01}$ | $\mathbf{0.20}^{\pm0.01}$ |
| llama3.2-3b | $0.00^{\pm0.00}$ | $\mathbf{0.27}^{\pm0.04}$ | $0.08^{\pm0.01}$ | $\mathbf{0.23}^{\pm0.01}$ | $0.12^{\pm0.02}$ | $\mathbf{0.25}^{\pm0.02}$ |
| llama3.1-8b | $0.05^{\pm0.00}$ | $\mathbf{0.28}^{\pm0.01}$ | $0.19^{\pm0.05}$ | $\mathbf{0.33}^{\pm0.10}$ | $0.27^{\pm0.05}$ | $\mathbf{0.36}^{\pm0.09}$ |
| ministral-8b | $0.05^{\pm0.00}$ | $\mathbf{0.29}^{\pm0.04}$ | $0.12^{\pm0.04}$ | $\mathbf{0.27}^{\pm0.01}$ | $0.15^{\pm0.04}$ | $\mathbf{0.28}^{\pm0.01}$ |
| mistral-22b | $0.22^{\pm0.04}$ | $\mathbf{0.41}^{\pm0.01}$ | $0.41^{\pm0.06}$ | $\mathbf{0.45}^{\pm0.01}$ | $0.40^{\pm0.04}$ | $\mathbf{0.47}^{\pm0.02}$ |
| qwen2.5-0.5b | $0.00^{\pm0.00}$ | $\mathbf{0.14}^{\pm0.03}$ | $0.02^{\pm0.00}$ | $\mathbf{0.20}^{\pm0.03}$ | $0.05^{\pm0.01}$ | $\mathbf{0.22}^{\pm0.01}$ |
| qwen2.5-1.5b | $0.00^{\pm0.00}$ | $\mathbf{0.20}^{\pm0.00}$ | $0.05^{\pm0.01}$ | $\mathbf{0.22}^{\pm0.01}$ | $0.08^{\pm0.01}$ | $\mathbf{0.23}^{\pm0.00}$ |
| qwen2.5-3b | $0.01^{\pm0.00}$ | $\mathbf{0.29}^{\pm0.01}$ | $0.16^{\pm0.04}$ | $\mathbf{0.22}^{\pm0.02}$ | $0.19^{\pm0.02}$ | $\mathbf{0.28}^{\pm0.04}$ |
| qwen2.5-7b | $0.21^{\pm0.01}$ | $\mathbf{0.33}^{\pm0.00}$ | $0.31^{\pm0.01}$ | $\mathbf{0.32}^{\pm0.04}$ | $\mathbf{0.39}^{\pm0.01}$ | $0.35^{\pm0.01}$ |
| qwen2.5-14b | $0.18^{\pm0.01}$ | $\mathbf{0.29}^{\pm0.04}$ | $\mathbf{0.33}^{\pm0.04}$ | $0.31^{\pm0.04}$ | $\mathbf{0.36}^{\pm0.02}$ | $0.26^{\pm0.00}$ |

Table 3: Accuracy of LLMs as logical parsers across different model sizes and prompting strategies (0-shot, 2-shot, 5-shot) with unconstrained (Unc.) versus grammar-constrained (Con.) decoding on the FOLIO datasets. As highlighted in Section 4.3, Accuracy $\leq$ Executable Rate in Table 5. We may achieve zero Accuracy if all the symbolic representations were invalid.

| | **GSM-symbolic** | | | | | |
|---|---|---|---|---|---|---|
| | **0shots** | | **2shots** | | **5shots** | |
| **Model** | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* |
| gemma2-2b | $0.00^{\pm0.00}$ | $\mathbf{0.15}^{\pm0.00}$ | $0.18^{\pm0.01}$ | $\mathbf{0.20}^{\pm0.00}$ | $0.18^{\pm0.01}$ | $\mathbf{0.21}^{\pm0.01}$ |
| gemma2-9b | $0.17^{\pm0.00}$ | $\mathbf{0.25}^{\pm0.01}$ | $\mathbf{0.44}^{\pm0.00}$ | $0.39^{\pm0.00}$ | $\mathbf{0.41}^{\pm0.05}$ | $0.37^{\pm0.03}$ |
| gemma2-27b | $\mathbf{0.31}^{\pm0.01}$ | $0.30^{\pm0.00}$ | $\mathbf{0.54}^{\pm0.00}$ | $0.49^{\pm0.00}$ | $\mathbf{0.51}^{\pm0.02}$ | $0.49^{\pm0.00}$ |
| llama3.2-1b | $0.00^{\pm0.00}$ | $\mathbf{0.03}^{\pm0.01}$ | $\mathbf{0.01}^{\pm0.00}$ | $0.02^{\pm0.00}$ | $0.01^{\pm0.00}$ | $\mathbf{0.03}^{\pm0.01}$ |
| llama3.2-3b | $0.00^{\pm0.00}$ | $\mathbf{0.12}^{\pm0.01}$ | $0.13^{\pm0.00}$ | $\mathbf{0.18}^{\pm0.00}$ | $0.16^{\pm0.04}$ | $\mathbf{0.19}^{\pm0.01}$ |
| llama3.1-8b | $0.00^{\pm0.00}$ | $\mathbf{0.27}^{\pm0.01}$ | $0.30^{\pm0.02}$ | $\mathbf{0.37}^{\pm0.02}$ | $0.28^{\pm0.01}$ | $\mathbf{0.35}^{\pm0.05}$ |
| ministral-8b | $0.01^{\pm0.01}$ | $\mathbf{0.12}^{\pm0.01}$ | $0.26^{\pm0.01}$ | $\mathbf{0.27}^{\pm0.01}$ | $0.26^{\pm0.01}$ | $\mathbf{0.28}^{\pm0.01}$ |
| mistral-22b | $0.00^{\pm0.00}$ | $\mathbf{0.13}^{\pm0.01}$ | $\mathbf{0.42}^{\pm0.01}$ | $0.38^{\pm0.01}$ | $\mathbf{0.42}^{\pm0.00}$ | $0.39^{\pm0.01}$ |
| qwen2.5-0.5b | $0.00^{\pm0.00}$ | $\mathbf{0.01}^{\pm0.00}$ | $0.01^{\pm0.00}$ | $0.01^{\pm0.01}$ | $0.02^{\pm0.01}$ | $0.02^{\pm0.01}$ |
| qwen2.5-1.5b | $0.00^{\pm0.00}$ | $\mathbf{0.05}^{\pm0.00}$ | $0.06^{\pm0.00}$ | $\mathbf{0.08}^{\pm0.01}$ | $0.07^{\pm0.01}$ | $\mathbf{0.09}^{\pm0.01}$ |
| qwen2.5-3b | $0.00^{\pm0.00}$ | $\mathbf{0.09}^{\pm0.00}$ | $0.18^{\pm0.11}$ | $\mathbf{0.33}^{\pm0.01}$ | $0.17^{\pm0.11}$ | $\mathbf{0.31}^{\pm0.01}$ |
| qwen2.5-7b | $0.00^{\pm0.00}$ | $\mathbf{0.20}^{\pm0.01}$ | $0.44^{\pm0.06}$ | $\mathbf{0.45}^{\pm0.03}$ | $0.46^{\pm0.08}$ | $\mathbf{0.47}^{\pm0.01}$ |
| qwen2.5-14b | $0.29^{\pm0.01}$ | $\mathbf{0.37}^{\pm0.01}$ | $\mathbf{0.57}^{\pm0.01}$ | $0.38^{\pm0.03}$ | $\mathbf{0.56}^{\pm0.02}$ | $0.36^{\pm0.01}$ |

Table 4: Accuracy of LLMs as logical parsers across different model sizes and prompting strategies (0-shot, 2-shot, 5-shot) with unconstrained (Unc.) versus grammar-constrained (Con.) decoding on the GSM-symbolic dataset. As highlighted in Section 4.3, Accuracy $\leq$ Executable Rate in Table 6. We may achieve zero Accuracy if all the symbolic representations were invalid.

| | **FOLIO** | | | | | |
|---|---|---|---|---|---|---|
| | **0shots** | | **2shots** | | **5shots** | |
| **Model** | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* |
| gemma2-2b | 0.07$^{\pm 0.05}$ | **0.60**$^{\pm 0.14}$ | 0.19$^{\pm 0.08}$ | **0.52**$^{\pm 0.28}$ | 0.16$^{\pm 0.07}$ | **0.65**$^{\pm 0.00}$ |
| gemma2-9b | 0.41$^{\pm 0.03}$ | **0.90**$^{\pm 0.00}$ | 0.64$^{\pm 0.09}$ | **0.84**$^{\pm 0.09}$ | 0.73$^{\pm 0.00}$ | **0.83**$^{\pm 0.00}$ |
| gemma2-27b | 0.67$^{\pm 0.00}$ | **0.94**$^{\pm 0.01}$ | 0.74$^{\pm 0.00}$ | **0.92**$^{\pm 0.01}$ | 0.79$^{\pm 0.01}$ | **0.89**$^{\pm 0.01}$ |
| llama3.2-1b | 0.00$^{\pm 0.00}$ | **0.57**$^{\pm 0.02}$ | 0.00$^{\pm 0.00}$ | **0.43**$^{\pm 0.07}$ | 0.01$^{\pm 0.00}$ | **0.62**$^{\pm 0.00}$ |
| llama3.2-3b | 0.00$^{\pm 0.00}$ | **0.72**$^{\pm 0.01}$ | 0.19$^{\pm 0.03}$ | **0.59**$^{\pm 0.04}$ | 0.25$^{\pm 0.01}$ | **0.64**$^{\pm 0.01}$ |
| llama3.1-8b | 0.09$^{\pm 0.02}$ | **0.78**$^{\pm 0.03}$ | 0.38$^{\pm 0.05}$ | **0.77**$^{\pm 0.06}$ | 0.43$^{\pm 0.04}$ | **0.78**$^{\pm 0.03}$ |
| ministral-8b | 0.09$^{\pm 0.05}$ | **0.83**$^{\pm 0.00}$ | 0.32$^{\pm 0.04}$ | **0.76**$^{\pm 0.01}$ | 0.37$^{\pm 0.04}$ | **0.77**$^{\pm 0.02}$ |
| mistral-22b | 0.40$^{\pm 0.06}$ | **0.87**$^{\pm 0.03}$ | 0.72$^{\pm 0.05}$ | **0.86**$^{\pm 0.01}$ | 0.69$^{\pm 0.00}$ | **0.86**$^{\pm 0.04}$ |
| qwen2.5-0.5b | 0.00$^{\pm 0.00}$ | **0.40**$^{\pm 0.06}$ | 0.07$^{\pm 0.00}$ | **0.58**$^{\pm 0.05}$ | 0.13$^{\pm 0.02}$ | **0.65**$^{\pm 0.01}$ |
| qwen2.5-1.5b | 0.01$^{\pm 0.01}$ | **0.56**$^{\pm 0.00}$ | 0.14$^{\pm 0.02}$ | **0.58**$^{\pm 0.06}$ | 0.18$^{\pm 0.01}$ | **0.58**$^{\pm 0.03}$ |
| qwen2.5-3b | 0.04$^{\pm 0.01}$ | **0.75**$^{\pm 0.01}$ | 0.29$^{\pm 0.04}$ | **0.54**$^{\pm 0.01}$ | 0.37$^{\pm 0.04}$ | **0.65**$^{\pm 0.06}$ |
| qwen2.5-7b | 0.37$^{\pm 0.04}$ | **0.72**$^{\pm 0.01}$ | 0.60$^{\pm 0.02}$ | **0.67**$^{\pm 0.07}$ | 0.64$^{\pm 0.01}$ | **0.73**$^{\pm 0.01}$ |
| qwen2.5-14b | 0.30$^{\pm 0.08}$ | **0.71**$^{\pm 0.03}$ | 0.62$^{\pm 0.04}$ | **0.72**$^{\pm 0.08}$ | **0.65**$^{\pm 0.00}$ | 0.62$^{\pm 0.02}$ |

Table 5: Executable Rate of LLMs as logical parsers across different model sizes and prompting strategies (0-shot, 2-shot, 5-shot) with unconstrained (Unc.) versus grammar-constrained (Con.) decoding on the FOLIO datasets.

| | **GSM-symbolic** | | | | | |
|---|---|---|---|---|---|---|
| | **0shots** | | **2shots** | | **5shots** | |
| **Model** | *Unc.* | *Con.* | *Unc.* | *Con.* | *Unc.* | *Con.* |
| gemma2-2b | 0.00$^{\pm 0.00}$ | **1.00**$^{\pm 0.01}$ | 0.78$^{\pm 0.02}$ | **1.00**$^{\pm 0.01}$ | 0.76$^{\pm 0.00}$ | **1.00**$^{\pm 0.00}$ |
| gemma2-9b | 0.43$^{\pm 0.02}$ | **1.00**$^{\pm 0.01}$ | 0.93$^{\pm 0.01}$ | **0.99**$^{\pm 0.00}$ | 0.93$^{\pm 0.01}$ | **0.99**$^{\pm 0.00}$ |
| gemma2-27b | 0.64$^{\pm 0.01}$ | **0.99**$^{\pm 0.00}$ | 0.96$^{\pm 0.00}$ | **1.00**$^{\pm 0.00}$ | 0.96$^{\pm 0.01}$ | **1.00**$^{\pm 0.00}$ |
| llama3.2-1b | 0.00$^{\pm 0.00}$ | **0.98**$^{\pm 0.00}$ | 0.27$^{\pm 0.01}$ | **0.98**$^{\pm 0.01}$ | 0.24$^{\pm 0.03}$ | **0.98**$^{\pm 0.01}$ |
| llama3.2-3b | 0.00$^{\pm 0.00}$ | **0.99**$^{\pm 0.01}$ | 0.70$^{\pm 0.02}$ | **1.00**$^{\pm 0.01}$ | 0.76$^{\pm 0.07}$ | **1.00**$^{\pm 0.01}$ |
| llama3.1-8b | 0.00$^{\pm 0.00}$ | **0.99**$^{\pm 0.00}$ | 0.76$^{\pm 0.08}$ | **1.00**$^{\pm 0.01}$ | 0.76$^{\pm 0.08}$ | **1.00**$^{\pm 0.01}$ |
| ministral-8b | 0.02$^{\pm 0.00}$ | **0.99**$^{\pm 0.00}$ | 0.83$^{\pm 0.00}$ | **1.00**$^{\pm 0.00}$ | 0.83$^{\pm 0.01}$ | **1.00**$^{\pm 0.01}$ |
| mistral-22b | 0.00$^{\pm 0.00}$ | **0.99**$^{\pm 0.00}$ | 0.93$^{\pm 0.00}$ | **1.00**$^{\pm 0.01}$ | 0.93$^{\pm 0.01}$ | **1.00**$^{\pm 0.00}$ |
| qwen2.5-0.5b | 0.00$^{\pm 0.00}$ | **0.94**$^{\pm 0.01}$ | 0.58$^{\pm 0.03}$ | **0.98**$^{\pm 0.01}$ | 0.53$^{\pm 0.10}$ | **0.98**$^{\pm 0.01}$ |
| qwen2.5-1.5b | 0.01$^{\pm 0.01}$ | **0.97**$^{\pm 0.01}$ | 0.65$^{\pm 0.02}$ | **0.99**$^{\pm 0.00}$ | 0.69$^{\pm 0.04}$ | **0.97**$^{\pm 0.03}$ |
| qwen2.5-3b | 0.00$^{\pm 0.00}$ | **0.97**$^{\pm 0.00}$ | 0.45$^{\pm 0.30}$ | **0.98**$^{\pm 0.01}$ | 0.46$^{\pm 0.28}$ | **0.98**$^{\pm 0.00}$ |
| qwen2.5-7b | 0.01$^{\pm 0.01}$ | **0.96**$^{\pm 0.01}$ | 0.83$^{\pm 0.09}$ | **1.00**$^{\pm 0.00}$ | 0.87$^{\pm 0.15}$ | **1.00**$^{\pm 0.00}$ |
| qwen2.5-14b | 0.59$^{\pm 0.01}$ | **1.00**$^{\pm 0.01}$ | 0.95$^{\pm 0.00}$ | **0.99**$^{\pm 0.00}$ | 0.96$^{\pm 0.01}$ | **0.99**$^{\pm 0.01}$ |

Table 6: Executable Rate of LLMs as logical parsers across different model sizes and prompting strategies (0-shot, 2-shot, 5-shot) with unconstrained (Unc.) versus grammar-constrained (Con.) decoding on the GSM-symbolic datasets.