Grammar Constrained Large Language Models for Logical Reasoning

1st Federico Raspanti
Eindhoven University of Technology
Eindhoven, The Netherlands
Université Côte d'Azur
Nice, France
f.raspanti@student.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 (NL) statements into symbolic representations, which are then verified by an external symbolic solver. However, ensuring the syntactic correctness of the symbolic representation remains challenging. In this work, we introduce GCLLM (Grammar-Constrained Logic Language Model), a method to enhance both the semantic and syntactic accuracy of symbolic representations generated by LLMs. Our approach leverages dynamically selected, semantically similar few-shot examples in the prompt and constrains the LLM output to adhere to a specific grammar. We also implement a self-verification process that utilizes open-source LLMs to correct errors in the generated symbolic representations. Empirical results indicate that GCLLM achieves a 21% improvement in syntactic accuracy compared to existing frameworks. Moreover GCLLM's design significantly reduces the likelihood of failures due to noncompliance of the sketcher, an issue observed in competing models such as LogicLM. However, no improvement in semantic accuracy was observed, with Chain-of-Thought (CoT) prompting remaining the most effective method for maintaining semantic integrity. These findings suggest that while GCLLM enhances syntactic robustness, CoT continues to be superior for ensuring semantic correctness in logical reasoning tasks.

CONTENTS

I. EXPERIMENTS

In this section, we illustrate the experiments we conducted to evaluate our framework. To account for the stochastic nature of LLM outputs, we run multiple trials for each configuration, reporting mean scores and standard deviations.

A. Baselines

1) End-to-end reasoning with LLM: We leverage LLMs to perform direct reasoning on the NL problems without any intermediate logical representation. The process is as follows:

1) Input: We provide the LLM with the NL rules and question from the dataset, as well as some examples.
2) Prompting: We instruct the LLM to analyze the given information and determine whether the statement in question is True, False, or Uncertain based solely on the provided rules.
3) Output: The LLM generates a direct answer (True, False, or Uncertain) along with a brief explanation of its reasoning.

2) Logic-LM: We implement the work of Pan et al. [**?**], Logic-LM, which does not make use of either Example Retrieval or GCD. We compare the performance of their framework to ours in order to assess the impact of these techniques.

B. Retrieving Relevant Examples

To investigate the impact of example selection on the performance of our pipeline, we implement and compare two

approaches for providing few-shot examples to the language model:

1) Static Examples: In this approach, we select a fixed set of examples from the training set that are used for all test samples. These examples are chosen to cover a range of logical structures and problem types representative of the dataset.

2) Dynamic Examples: This method involves selecting examples tailored to each specific test sample. We implement this approach as described in **??**.

To ensure a fair comparison with our baseline Logic-LM, we follow their implementation by providing 2 examples in the prompt, for each sample.

To quantify the impact of dynamic example selection, we compare the semantic accuracy [Sec. **??**] of the generated symbolic representations for both static and dynamic approaches. This comparison allows us to assess whether tailoring examples to each test sample leads to improved performance in logical reasoning tasks.

C. Grammar-Constrained Self-verification

To evaluate our grammar-constrained self-verification approach, we implement it using various open-source LLMs with logit access, constrained by our dynamically updated Context-Free Grammar (CFG).

We use the process described in equation **??**, where the LLM's generation is constrained by the grammar G, which is dynamically updated based on the predicates extracted from the initial sketch.

D. Fine-tuned Self-verification

We also evaluate the impact of fine-tuning on self-verification performance. To do so, we propose **FOLIO-Refinement**, a synthetic dataset created by introducing syntax error into the FOL formulas of the FOLIO dataset.

After fine-tuning the OS LLMs, we integrate them into our self-verification pipeline, replacing their non-fine-tuned counterparts. We then compared the performance of the fine-tuned models against both the non-fine-tuned versions and the unconstrained self-verification approach.

E. Self-verification with Backup

Finally, we introduce an additional strategy to handle samples that remain unexecutable after the self-verification process. Instead of assigning these samples the control label as described in Sec. **??**, we adopt an approach similar to LogicLM [**?**]. For these cases, we directly employ an LLM to generate the answer to the logical problem.

To implement this strategy we promp the LLM with the original natural language problem and question, following the approach described in Sec. I-A1. We then use the LLM's output as the final answer for samples that could not be processed through our main pipeline.

Our aim is to provide a response for all input samples, potentially improving the overall performance of our system. This approach also enables us to compare the effectiveness of our logical reasoning pipeline against direct LLM inference on challenging cases.

We measure the performance of the experiments in Sections I-C - I-E in terms of syntactic validity [Sec. **??**] and semantic accuracy [Sec. **??**] of the generated symbolic representations.

II. Evaluation

III. Results and discussion

We report the results of our experiments on dynamic example retrieval [**??**] in Table **??**.

We also report the results of the Self-verification experiments[Sec. I-C-I-E], as well as our end-to-end and LogicLM baselines [Sec. I-A1, I-A2] on the FOLIO and LogicNLI datasets in Table I and II respectively. The Chain-of-Thought strategy does not use a symbolic solver nor backup, therefore we do not report Weighted F1 with backup and % of executable samples (N/A).

Finally, we report the results of backup-specific experiments in Table IV for FOLIO, and in Table V for LogicNLI.

A. Dynamic Examples Retrieval

The dynamic example selection approach [Tab. **??**] showed that selecting the few-shot examples dynamically, according to embedding similarity with the test sample, improves the semantic accuracy of the generated symbolic problems by $1-2\%$ when using GPT-4o, and degrades it by 3% when using GPT-3.5-Turbo. When considering only the samples that the symbolic solver can execute (executable samples), we observe that the changes in weighted F1 score follow closely the changes when considering all samples. Moreover, our dynamic approach achieves improvements in syntactic accuracy (% of executable samples) of up to 6.41% on FOLIO, with GPT-4o.

These results seems to suggest that when it comes to the semantic accuracy of the logical formulas from Natural Language statements, **these models are fairly agnostic to the semantic similarity of the few-shot-examples with the test problem** [Sec. **??**], and the performance relies on the underlying capabilities of the language model. Conversely, **dynamic examples seem to improve the syntactic accuracy** of the generate symbolic formulas.

B. LogicLM

When reproducing the results of LogicLM, we observed that the weighted F1 score and the fraction of executable samples were exceptionally low for LogicNLI, and for FOLIO when using GPT-4o as the sketcher (Tab. I and II, results in red). The only instance where the performance was comparable to GCLLM and CoT was on FOLIO, when using GPT-3.5-Turbo as the sketcher, which is one of the configurations of the original paper [**?**].

Sketcher	Strategy	Refiner	Weighted F1 (all samples)	Weighted F1 (all samples, with backup)	Weighted F1 (executable samples)	Executable Samples (%)
GPT-3.5-Turbo	CoT	GPT-3.5-Turbo	$\mathbf{0.52^{\pm 0.0}}$	N/A		
	LogicLM	GPT-3.5-Turbo	$0.49^{\pm 0.03}$	$0.62^{\pm 0.01}$	$\mathbf{0.63^{\pm 0.03}}$	$62.75^{\pm 0}\%$
	GCLLM	Llama 2 7b	$0.5^{\pm 0.02}$	$0.62^{\pm 0.03}$	$0.6^{\pm 0.03}$	$72.58^{\pm 1}\%$
		Llama 2 7b ♦	$0.5^{\pm 0.01}$	$0.62^{\pm 0.02}$	$0.61^{\pm 0.02}$	$71.43^{\pm 2}\%$
		Llama 2 7b ★	$0.5^{\pm 0.01}$	$0.58^{\pm 0.04}$	$0.56^{\pm 0.02}$	$\mathbf{83.58^{\pm 2}}\%$
		Llama 2 7b ♦ ★	$0.5^{\pm 0.02}$	$0.58^{\pm 0.04}$	$0.55^{\pm 0.03}$	$83.25^{\pm 2}\%$
		Llama 2 13b	$0.5^{\pm 0.01}$	$0.62^{\pm 0.02}$	$0.61^{\pm 0.02}$	$70.94^{\pm 1}\%$
		Llama 2 13b ♦	$0.5^{\pm 0.01}$	$\mathbf{0.63^{\pm 0.03}}$	$0.61^{\pm 0.02}$	$71.92^{\pm 1}\%$
		Llama 2 13b ★	$0.51^{\pm 0.01}$	$0.6^{\pm 0.03}$	$0.57^{\pm 0.02}$	$81.28^{\pm 2}\%$
		Llama 2 13b ♦ ★	$0.5^{\pm 0.02}$	$0.58^{\pm 0.03}$	$0.55^{\pm 0.03}$	$\mathbf{83.58^{\pm 2}}\%$
		Llama 3 8b	$0.5^{\pm 0.02}$	$0.62^{\pm 0.02}$	$0.61^{\pm 0.03}$	$71.76^{\pm 1}\%$
		Llama 3 8b ♦	$0.5^{\pm 0.01}$	$0.61^{\pm 0.02}$	$0.61^{\pm 0.02}$	$71.76^{\pm 1}\%$
		Llama 3 8b ★	$0.51^{\pm 0.01}$	$0.59^{\pm 0.04}$	$0.57^{\pm 0.02}$	$83.42^{\pm 2}\%$
		Llama 3 8b ♦ ★	$0.5^{\pm 0.01}$	$0.61^{\pm 0.01}$	$0.59^{\pm 0.01}$	$76.85^{\pm 0}\%$
GPT-4o	CoT	GPT-4o	$\mathbf{0.71^{\pm 0.0}}$	N/A		
	LogicLM	GPT-4o	$0.06^{\pm 0.0}$	$0.71^{\pm 0.0}$	$0.84^{\pm 0.0}$	$3.43^{\pm}\%$
	GCLLM	Llama 2 7b	$0.67^{\pm 0.01}$	$0.73^{\pm 0.01}$	$0.72^{\pm 0.01}$	$87.85^{\pm 0}\%$
		Llama 2 7b ♦	$0.67^{\pm 0.01}$	$0.74^{\pm 0.0}$	$0.72^{\pm 0.01}$	$87.85^{\pm 0}\%$
		Llama 2 7b ★	$0.68^{\pm 0.01}$	$0.73^{\pm 0.02}$	$0.71^{\pm 0.01}$	$91.95^{\pm 1}\%$
		Llama 2 7b ♦ ★	$0.69^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.72^{\pm 0.0}$	$\mathbf{93.92^{\pm 0}}\%$
		Llama 2 13b	$0.68^{\pm 0.01}$	$0.74^{\pm 0.01}$	$0.73^{\pm 0.0}$	$87.36^{\pm 1}\%$
		Llama 2 13b ♦	$0.68^{\pm 0.01}$	$0.74^{\pm 0.01}$	$0.73^{\pm 0.0}$	$87.19^{\pm 0}\%$
		Llama 2 13b ★	$0.68^{\pm 0.0}$	$0.73^{\pm 0.01}$	$0.71^{\pm 0.01}$	$92.78^{\pm 1}\%$
		Llama 2 13b ♦ ★	$0.68^{\pm 0.01}$	$0.73^{\pm 0.01}$	$0.72^{\pm 0.01}$	$92.28^{\pm 1}\%$
		Llama 3 8b	$0.68^{\pm 0.01}$	$0.74^{\pm 0.01}$	$0.73^{\pm 0.0}$	$87.68^{\pm 1}\%$
		Llama 3 8b ♦	$0.67^{\pm 0.01}$	$0.74^{\pm 0.01}$	$0.73^{\pm 0.01}$	$87.36^{\pm 1}\%$
		Llama 3 8b ★	$0.67^{\pm 0.01}$	$0.71^{\pm 0.02}$	$0.7^{\pm 0.02}$	$93.27^{\pm 0}\%$
		Llama 3 8b ♦ ★	$0.67^{\pm 0.01}$	$0.73^{\pm 0.01}$	$0.71^{\pm 0.01}$	$91.3^{\pm 1}\%$

TABLE I

RESULTS AFTER 3 ROUNDS OF SELF-VERIFICATION ON THE FOLIO DATASET. IN RED, EXPERIMENTS WHERE THE % OF EXECUTABLE SAMPLES WAS EXCEPTIONALLY LOW.
★: GRAMMAR-CONSTRAINED MODEL
♦: FINE-TUNED MODEL

Upon further investigation, we found that this was due to the sketchers not properly following the prompt instruction [Lst. **??**], and spontaneously generating an unrequested explanation [Lst. **??**], making the response impossible to parse by the LogicLM framework. We suspect that this is because LogicLM relies on the model strictly following the examples provided, without explicitly requesting a response format. This limits the generality of LogicLM, as the prompt must always be tested and tuned on the specific sketcher, to increase the likelihood that it will respect the instructions.

In contrast, the prompts of GCLLM specifically request that the response is given in JSON format [Sec. **??**], which makes the sketcher much more compliant with the expected response format.

C. Grammar-constrained self-verification

The impact of Grammar-Constrained self-verification can be seen by comparing the rows with and without the ★ symbol in Tables I and II. When taking into consideration all samples, for both the FOLIO and LogicNLI datasets, the weighted F1 score with each grammar-constrained refiner (llama-2-7b, llama-2-13b and llama-3-8b) is virtually equivalent to weighted F1 of its unconstrained version, with improvements of up to 0.01. If we take into consideration only the executable samples, this still holds for the LogicNLI dataset when using both the GPT-3.5-Turbo and GPT-4o sketchers, and for the FOLIO dataset when using the GPT-4o sketcher. When using the GPT-3.5-Turbo sketcher on the FOLIO dataset, we see a slight degradation in weighted F1 for the grammar-constrained refiners, of up to -0.06. These results suggest that **GCD does not increase semantic accuracy**.

However, we observe that when using GCD the % of executable samples significantly increases, by up to 11% when using GPT-3.5-Turbo sketcher with llama-2-7b refiner on FOLIO , and up to 7.5 % when using GPT-4o with llama-2-7b on LogicNLI, which shows that **GCD is effective for increasing the syntactic accuracy**.

Figure 1 shows the number of samples that were corrected uniquely by one refiner, compared to the number of samples that were corrected commonly by all refiners. We observe that for both dataset, the vast majority of samples were corrected commonly by all three refiners. This, together with the fact that the models perform similarly both in terms of semantic and syntactic accuracy, suggests that **the refiners we take into consideration have similar performance for GCLLM**. This is likely due to the fact that the refiners we selected have similar architectures and number of parameters.

We investigate why the increase in syntactic accuracy when using a grammar-constrained refiner does not correspond to an increase in semantic accuracy by analyzing the samples

Sketcher	Strategy	Refiner	Weighted F1 (all samples)	Weighted F1 (all samples, with backup)	Weighted F1 (executable samples)	Executable Samples (%)
GPT-3.5-Turbo	CoT	GPT-3.5-Turbo	$\mathbf{0.36}^{\pm 0.0}$	N/A		
	LogicLM	GPT-3.5-Turbo	$0.0^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.0}$	$0.4^{\pm}\%$
	GCLLM	Llama 2 7b	$0.21^{\pm 0.0}$	$0.26^{\pm 0.01}$	$0.23^{\pm 0.0}$	$84.89^{\pm 0}\%$
		Llama 2 7b ♦	$0.22^{\pm 0.0}$	$0.26^{\pm 0.01}$	$0.23^{\pm 0.0}$	$88.89^{\pm 0}\%$
		Llama 2 7b ★	$0.21^{\pm 0.0}$	$0.24^{\pm 0.0}$	$0.22^{\pm 0.0}$	$92.71^{\pm 0}\%$
		Llama 2 7b ♦ ★	$0.22^{\pm 0.0}$	$0.24^{\pm 0.0}$	$0.22^{\pm 0.0}$	$92.69^{\pm 0}\%$
		Llama 2 13b	$0.21^{\pm 0.0}$	$0.26^{\pm 0.01}$	$0.23^{\pm 0.0}$	$84.89^{\pm 0}\%$
		Llama 2 13b ♦	$0.22^{\pm 0.0}$	$0.26^{\pm 0.01}$	$0.23^{\pm 0.01}$	$88.42^{\pm 0}\%$
		Llama 2 13b ★	$0.22^{\pm 0.0}$	$0.24^{\pm 0.0}$	$0.22^{\pm 0.0}$	$92.73^{\pm 0}\%$
		Llama 2 13b ♦ ★	$0.22^{\pm 0.0}$	$0.24^{\pm 0.0}$	$0.22^{\pm 0.0}$	$92.67^{\pm 0}\%$
		Llama 3 8b	$0.21^{\pm 0.0}$	$0.26^{\pm 0.01}$	$0.23^{\pm 0.0}$	$84.96^{\pm 0}\%$
		Llama 3 8b ♦	$0.22^{\pm 0.0}$	$0.26^{\pm 0.01}$	$0.23^{\pm 0.0}$	$88.53^{\pm 0}\%$
		Llama 3 8b ★	$0.22^{\pm 0.0}$	$0.24^{\pm 0.0}$	$0.22^{\pm 0.0}$	$\mathbf{92.78}^{\pm 0}\%$
		Llama 3 8b ♦ ★	$0.22^{\pm 0.0}$	$0.24^{\pm 0.0}$	$0.22^{\pm 0.0}$	$92.76^{\pm 0}\%$
GPT-4o	CoT	GPT-4o	$\mathbf{0.73}^{\pm 0.0}$	N/A		
	LogicLM	GPT-4o	$0.0^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.0^{\pm 0.0}$	$0.0^{\pm 0.0}$
	GCLLM	Llama 2 7b	$0.46^{\pm 0.0}$	$0.5^{\pm 0.0}$	$0.48^{\pm 0.0}$	$92.46^{\pm 0}\%$
		Llama 2 7b ♦	$0.46^{\pm 0.0}$	$0.5^{\pm 0.0}$	$0.48^{\pm 0.0}$	$93.55^{\pm 0}\%$
		Llama 2 7b ★	$0.47^{\pm 0.0}$	$0.48^{\pm 0.0}$	$0.47^{\pm 0.0}$	$\mathbf{99.0}^{\pm 0}\%$
		Llama 2 7b ♦ ★	$0.47^{\pm 0.0}$	$0.48^{\pm 0.0}$	$0.47^{\pm 0.0}$	$\mathbf{99.0}^{\pm 0}\%$
		Llama 2 13b	$0.46^{\pm 0.0}$	$0.5^{\pm 0.0}$	$0.48^{\pm 0.0}$	$92.46^{\pm 0}\%$
		Llama 2 13b ♦	$0.46^{\pm 0.0}$	$0.49^{\pm 0.0}$	$0.48^{\pm 0.0}$	$94.42^{\pm 0}\%$
		Llama 2 13b ★	$0.47^{\pm 0.0}$	$0.48^{\pm 0.0}$	$0.47^{\pm 0.0}$	$\mathbf{99.0}^{\pm 0}\%$
		Llama 2 13b ♦ ★	$0.47^{\pm 0.0}$	$0.48^{\pm 0.0}$	$0.47^{\pm 0.0}$	$\mathbf{99.0}^{\pm 0}\%$
		Llama 3 8b	$0.46^{\pm 0.0}$	$0.5^{\pm 0.0}$	$0.48^{\pm 0.0}$	$92.46^{\pm 0}\%$
		Llama 3 8b ♦	$0.46^{\pm 0.0}$	$0.49^{\pm 0.0}$	$0.48^{\pm 0.0}$	$94.13^{\pm 0}\%$
		Llama 3 8b ★	$0.47^{\pm 0.0}$	$0.47^{\pm 0.0}$	$0.47^{\pm 0.0}$	$\mathbf{99.0}^{\pm 0}\%$
		Llama 3 8b ♦ ★	$0.47^{\pm 0.0}$	$0.48^{\pm 0.0}$	$0.47^{\pm 0.0}$	$\mathbf{99.0}^{\pm 0}\%$

TABLE II

RESULTS AFTER 3 ROUNDS OF SELF-VERIFICATION ON THE LOGICNLI DATASET. IN RED, EXPERIMENTS WHERE THE % OF EXECUTABLE SAMPLES WAS EXCEPTIONALLY LOW.
★: GRAMMAR-CONSTRAINED MODEL
♦: FINE-TUNED MODEL

		FOLIO	LogicNLI
Sketcher	**Refiner**	**Weighted F1 (corrected samples)**	**Weighted F1 (corrected samples)**
GPT-3.5-Turbo	Random	$0.3^{\pm 0.03}(25)$	$0.34^{\pm 0.01}(119)$
	Llama 2 7b ★	$0.23^{\pm 0.02}(25)$	$0.15^{\pm 0.03}(118)$
	Llama 2 13b ★	$0.23^{\pm 0.02}(27)$	$0.14^{\pm 0.03}(119)$
	Llama 3 8b ★	$0.38^{\pm 0.13}(14)$	$0.15^{\pm 0.02}(119)$
GPT-4o	Random	$0.35^{\pm 0.04}(203)$	$0.33^{\pm 0.02}(1499)$
	Llama 2 7b ★	$0.47^{\pm 0.12}(16)$	$0.33^{\pm 0.01}(98)$
	Llama 2 13b ★	$0.38^{\pm 0.22}(13)$	$0.32^{\pm 0.0}(98)$
	Llama 3 8b ★	$0.36^{\pm 0.18}(11)$	$0.34^{\pm 0.0}(98)$

TABLE III

WEIGHTED F1 SCORE OF THE SAMPLES WHOSE SYNTAX WAS CORRECTED BY THE REFINER. IN PARENTHESIS, THE AVERAGE NUMBER OF SAMPLES AVERAGED ACROSS THREE EXPERIMENTS, ROUNDED DOWN.

that are not executable when using the unconstrained refiner (i.e. the refiner without a star), but that are executable when using the grammar constrained refiner (★). Table III shows the weighted F1 score of those samples, along with the weighted F1 of a random refiner, which picks an answer (True, False or Uncertain) at random and assigns it to the sample. For both datasets, the weighted F1 score is **lower than the random refiner** when using GPT-3.5-Turbo as the sketcher, and **equivalent to the random refiner** when using GPT-4o. We suspect that this is caused by the refiners changing the meaning of the symbolic formulas when attempting to correct their syntax.

In order to verify this, we go through each of the samples that are executable only with the grammar-constrained refiner, and manually refine the original invalid formulas. In doing so, we attempt to preserve the meaning of the original formulas, only fixing syntax errors. Listing [**??**] shows an example of an original invalid formula, the refiner-generated correction, and the manually-generated correction.

Figure 2 shows the distribution of the actual labels, compared to the distribution of the predicted answer with the manually corrected samples (Manual) and the samples corrected by the grammar-constrained refiner (★). Overall, **manual and grammar-constrained self verification yield very similar labels** for both datasets, as the distributions of the predicted labels are very close. This suggests that the lack of improvement

Fig. 1. Number of unique errors for each refiner compared to the number of common errors

in semantic accuracy is not related to the the refiner changing the meaning of the corrected symbolic formulas, but rather to the whole sketcher-generated symbolic problems not correctly encapsulating the meaning of the original logical problems.

D. Fine-tuned self-verification

The results of fine-tuning on the FOLIO-Refinement dataset [Sec. **??**] are indicated by the ♦ symbol in Tables I and II. Just as for the grammar-constrained models, the weighted F1 for the fine-tuned refiners is almost equivalent to their unconstrained counterparts. Moreover, the % of executable samples does not seems to improve with fine-tuning. These results suggest that **fine-tuning does not improve neither syntactic or semantic accuracy**.

E. Combining fine-tuning and GCD

When combining grammar constraints with fine-tuning (★♦), the weighted F1 and % of executable samples are very similar to the grammar constrained versions (★). The only exception is when inferring on FOLIO with GPT-3.5-Turbo as the sketcher and llama-3-8b as the refiner, where we see a drop of 7% in the percentage of executable samples. These results confirm the notion that **fine-tuning the open-source refiners does not lead to any improvement in semantic or syntactic accuracy**.

Table III shows the weighted F1 score on the samples that are not executable when using the unconstrained refiner, but that are executable when using the grammar constrained refiner (♦★). The results are in line with the discussion in III-C.

F. Self-verification with backup

The results for our experiments with backup are shown in Tables I and IV for the FOLIO dataset, and in Tables II and V for the LogicNLI dataset. In general, we observe that if the symbolic solver could not run the generated symbolic problem, asking a black-box LLM to answer the question directly improves the **semantic accuracy** (Tables I and II, "all samples, with backup"). The magnitude of this improvement seems to be both dataset and model dependent, with improvements as high as 0.12 on FOLIO, with GPT-3.5-Turbo as the sketcher, and as low as 0.01 on LogicNLI, with GPT-4o as the sketcher.

To better assess the performance of the CoT backup against the performance of GCLLM, we report in Tables IV and V the Weighted F1 score of GCLLM on the samples that the symbolic solver can execute, against the Weighted F1 of the CoT backup on the same samples. We observe that GCLLM has better performance on the executable samples than the CoT backup on the FOLIO dataset, with improvements of up to 0.12 when using GPT-3.5-Turbo as the sketcher and Llama 3 8b ♦ as the refiner. When using the GPT-4o sketcher, the improvement is not significant. We also observe that the CoT backup achieves better performance on the non-executable samples than on the executable samples, suggesting that, for the FOLIO dataset, **combining the two approaches (GCLLM for executable samples, CoT for the non-executable samples) leads to improved semantic accuracy.**

Coversely, when inferring on the LogicNLI dataset, we observe that the CoT backup achieves much higher scores than GCLLM, with improvements of up to 0.26 when using GPT-4o as the sketcher and Llama 2 7b as the refiner. Moreover, unlike for FOLIO, for LogicNLI the CoT backup seems to have slightly better performance on the executable samples than on the non-executable samples with GPT-3.5-Turbo as the sketcher, and virtually equivalent performance with GPT-4o as the sketcher. This suggests that, for LogicNLI, **CoT is the approach that maximizes semantic accuracy**.

Sketcher	Refiner	Weighted F1 (executable samples)	Weighted F1 of backup (executable samples)	Weighted F1 of backup (non-executable samples)
GPT-3.5-Turbo	Llama 2 7b	$0.6^{\pm 0.03}$	$0.5^{\pm 0.02}$	$0.58^{\pm 0.06}$
	Llama 2 7b ♦	$\mathbf{0.61^{\pm 0.02}}$	$0.5^{\pm 0.02}$	$0.57^{\pm 0.04}$
	Llama 2 7b ★	$0.56^{\pm 0.02}$	$\mathbf{0.52^{\pm 0.03}}$	$0.52^{\pm 0.16}$
	Llama 2 7b ♦ ★	$0.55^{\pm 0.03}$	$0.51^{\pm 0.02}$	$0.55^{\pm 0.11}$
	Llama 2 13b	$\mathbf{0.61^{\pm 0.02}}$	$0.5^{\pm 0.01}$	$0.56^{\pm 0.04}$
	Llama 2 13b ♦	$\mathbf{0.61^{\pm 0.02}}$	$0.49^{\pm 0.02}$	$0.59^{\pm 0.05}$
	Llama 2 13b ★	$0.57^{\pm 0.02}$	$0.5^{\pm 0.03}$	$\mathbf{0.62^{\pm 0.12}}$
	Llama 2 13b ♦ ★	$0.55^{\pm 0.03}$	$0.51^{\pm 0.02}$	$0.58^{\pm 0.07}$
	Llama 3 8b	$\mathbf{0.61^{\pm 0.03}}$	$0.5^{\pm 0.01}$	$0.57^{\pm 0.03}$
	Llama 3 8b ♦	$\mathbf{0.61^{\pm 0.02}}$	$0.51^{\pm 0.01}$	$0.54^{\pm 0.04}$
	Llama 3 8b ★	$0.57^{\pm 0.02}$	$\mathbf{0.52^{\pm 0.02}}$	$0.54^{\pm 0.11}$
	Llama 3 8b ♦ ★	$0.59^{\pm 0.01}$	$0.51^{\pm 0.01}$	$0.56^{\pm 0.03}$
GPT-4o	Llama 2 7b	$0.72^{\pm 0.01}$	$\mathbf{0.71^{\pm 0.01}}$	$0.79^{\pm 0.05}$
	Llama 2 7b ♦	$0.72^{\pm 0.01}$	$0.7^{\pm 0.01}$	$0.8^{\pm 0.03}$
	Llama 2 7b ★	$0.71^{\pm 0.01}$	$0.7^{\pm 0.01}$	$0.89^{\pm 0.05}$
	Llama 2 7b ♦ ★	$0.72^{\pm 0.0}$	$0.7^{\pm 0.0}$	$\mathbf{0.98^{\pm 0.04}}$
	Llama 2 13b	$\mathbf{0.73^{\pm 0.0}}$	$\mathbf{0.71^{\pm 0.01}}$	$0.79^{\pm 0.06}$
	Llama 2 13b ♦	$\mathbf{0.73^{\pm 0.0}}$	$0.7^{\pm 0.01}$	$0.8^{\pm 0.04}$
	Llama 2 13b ★	$0.71^{\pm 0.01}$	$0.7^{\pm 0.0}$	$\mathbf{0.98^{\pm 0.03}}$
	Llama 2 13b ♦ ★	$0.72^{\pm 0.01}$	$0.7^{\pm 0.0}$	$0.89^{\pm 0.03}$
	Llama 3 8b	$\mathbf{0.73^{\pm 0.0}}$	$0.7^{\pm 0.01}$	$0.8^{\pm 0.04}$
	Llama 3 8b ♦	$\mathbf{0.73^{\pm 0.01}}$	$0.7^{\pm 0.01}$	$0.81^{\pm 0.04}$
	Llama 3 8b ★	$0.7^{\pm 0.02}$	$\mathbf{0.71^{\pm 0.0}}$	$0.84^{\pm 0.03}$
	Llama 3 8b ♦ ★	$0.71^{\pm 0.01}$	$0.7^{\pm 0.0}$	$0.92^{\pm 0.07}$

TABLE IV

EXECUTABLE SAMPLES F1 SCORE AFTER 3 ROUNDS OF SELF-VERIFICATION ON THE FOLIO DATASET, COMPARED WITH THE F1 SCORE OF THE BACKUP ON BOTH EXECUTABLE AND NON-EXECUTABLE SAMPLES

★: GRAMMAR-CONSTRAINED MODEL

♦: FINE-TUNED MODEL

Sketcher	Refiner	Weighted F1 (executable samples)	Weighted F1 of backup (executable samples)	Weighted F1 of backup (non-executable samples)
GPT-3.5-Turbo	Llama 2 7b	$0.23^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.32^{\pm 0.02}$
	Llama 2 7b ♦	$0.23^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.02}$
	Llama 2 7b ★	$0.22^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.03}$
	Llama 2 7b ♦ ★	$0.22^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.03}$
	Llama 2 13b	$0.23^{\pm 0.0}$	$\mathbf{0.37^{\pm 0.0}}$	$0.32^{\pm 0.02}$
	Llama 2 13b ♦	$0.23^{\pm 0.01}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.01}$
	Llama 2 13b ★	$0.22^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.32^{\pm 0.02}$
	Llama 2 13b ♦ ★	$0.22^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.02}$
	Llama 3 8b	$0.23^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.32^{\pm 0.02}$
	Llama 3 8b ♦	$0.23^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.01}$
	Llama 3 8b ★	$0.22^{\pm 0.0}$	$0.36^{\pm 0.0}$	$0.33^{\pm 0.01}$
	Llama 3 8b ♦ ★	$0.22^{\pm 0.0}$	$0.36^{\pm 0.0}$	$\mathbf{0.34^{\pm 0.02}}$
GPT-4o	Llama 2 7b	$0.48^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.72^{\pm 0.0}$
	Llama 2 7b ♦	$0.48^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.72^{\pm 0.02}$
	Llama 2 7b ★	$0.47^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.74^{\pm 0.0}$
	Llama 2 7b ♦ ★	$0.47^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.74^{\pm 0.0}$
	Llama 2 13b	$0.48^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.72^{\pm 0.0}$
	Llama 2 13b ♦	$0.48^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.73^{\pm 0.01}$
	Llama 2 13b ★	$0.47^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.74^{\pm 0.0}$
	Llama 2 13b ♦ ★	$0.47^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.74^{\pm 0.0}$
	Llama 3 8b	$0.48^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.72^{\pm 0.0}$
	Llama 3 8b ♦	$0.48^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.73^{\pm 0.01}$
	Llama 3 8b ★	$0.47^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.74^{\pm 0.0}$
	Llama 3 8b ♦ ★	$0.47^{\pm 0.0}$	$0.73^{\pm 0.0}$	$0.74^{\pm 0.0}$

TABLE V

EXECUTABLE SAMPLES F1 SCORE AFTER 3 ROUNDS OF SELF-VERIFICATION ON THE LOGICNLI DATASET, COMPARED WITH THE F1 SCORE OF THE BACKUP ON BOTH EXECUTABLE AND NON-EXECUTABLE SAMPLES

★: GRAMMAR-CONSTRAINED MODEL

♦: FINE-TUNED MODEL

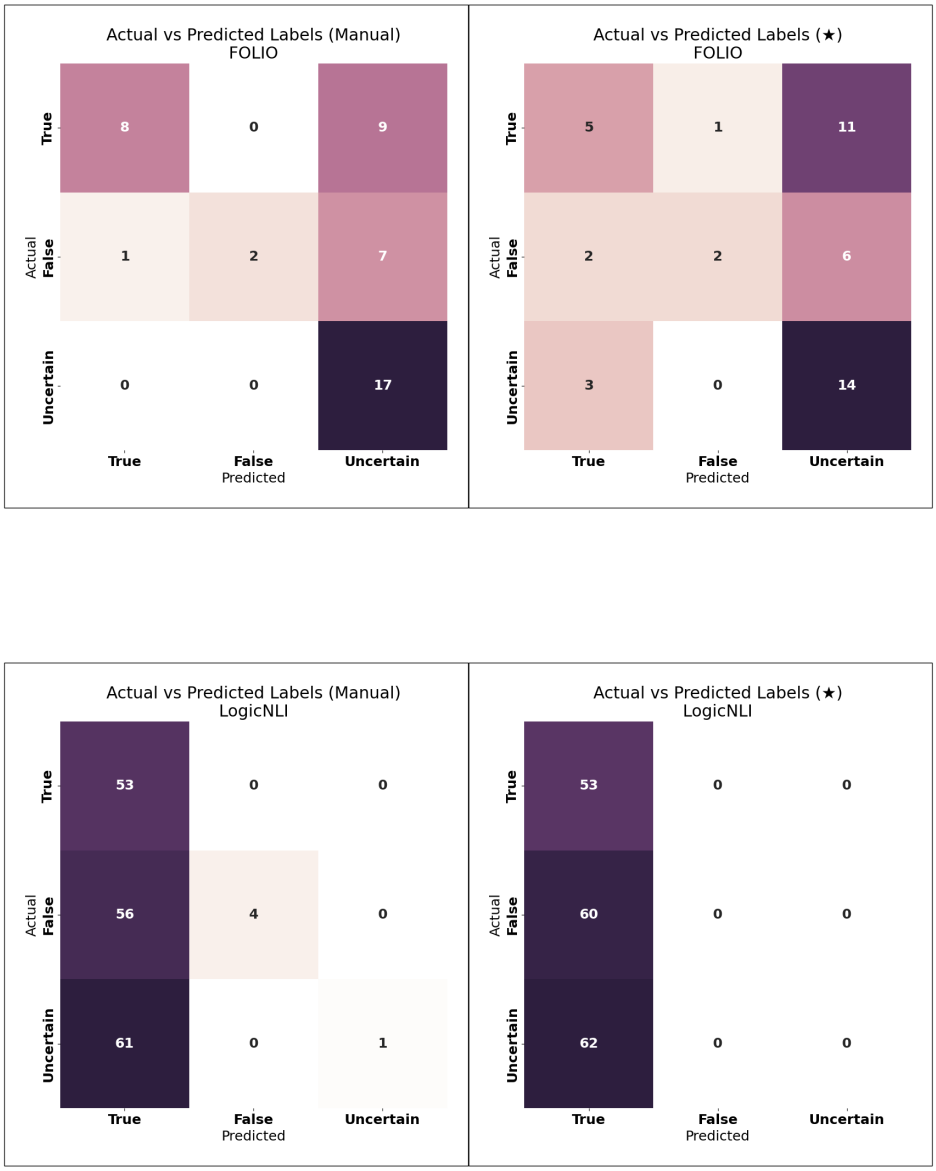

Fig. 2. Confusion matrix for samples that were fixed by the grammar-constrained refiner and manually. Starting from top-left to bottom right: manually-corrected samples of FOLIO, refiner-corrected samples of FOLIO, manually-corrected samples of LogicNLI and refiner-corrected samples of LogicNLI.

G. FOLIO vs LogicNLI

When comparing the results on the two datasets across all configurations, it appears that the **semantic accuracy** of the generated symbolic problems is much higher for FOLIO than for LogicNLI. We suspect that this is due to the fact that LogicNLI "effectively disentangles the target FOL reasoning from commonsense inference" [**?**], meaning that its natural language problems cannot be reasoned about using common sense, but only through rigorous, step-by-step logical reasoning. On the other hand, FOLIO does not take preemptive measures against the LLM's ability to infer the correct answer to a logical problem using common sense.

At the same time, we observe that the **syntactic accuracy** of the generated symbolic problems is higher for LogicNLI than it is for FOLIO. The reason for this is likely that LogicNLI's rules are less complex, containing only one or two logical relations, with one or two levels of nesting, while FOLIO's rules often contain more logical relations, and more levels of nesting. This makes it easier for an LLM to parse LogciNLI's natural language statements into symbolic

formulas with correct syntax.

IV. Conclusion and Future Work

In this paper, we introduced GCLLM, a framework for logical reasoning using large language models (LLMs) augmented with symbolic solvers, example retrieval and grammar-constrained self-verification. Our experiments on the FOLIO and LogicNLI datasets yielded several insights:

A. Dynamic Example Retrieval

Our simplified approach to retrieving relevant examples did not achieve better semantic accuracy compared to static example selection. However, it did lead to increased syntactic accuracy of the generated symbolic formulas, with improvements of up to 6.41% on the FOLIO dataset when using GPT-4o as the sketcher. This suggests that dynamic example selection may be beneficial for improving the structural correctness of LLM-generated logical formulas, even if it does not necessarily enhance their semantic accuracy.

B. GCLLM vs LogicLM

Our implementation proved to be more reliable than the implementation of LogicLM. By explicitly requesting responses in JSON format, we were able to achieve consistent performance across different LLM configurations, avoiding the issues of unparsable outputs that degraded the performance of the LogicLM framework.

C. Grammar-constrained self-verification

Our grammar-constrained verification approach successfully increased the syntactic accuracy of the generated symbolic formulas, with improvements in the percentage of executable samples of up to 11% on FOLIO and 7.5% on LogicNLI. However, this increase in syntactic accuracy did not translate to improved semantic accuracy. Our experiments suggest that this was due to the pre-refined sketches generated by the black-box LLMs not properly capturing the meaning of the logical problem. This indicates that future work should focus on improving the initial sketch generation to better encapsulate the semantics of the logical problems.

D. Fine-tuned self-verification

Our experiments with fine-tuning the open-source refiners on the FOLIO-Refinement dataset did not seem to impact the framework's performance significantly. Neither semantic accuracy nor syntactic accuracy showed notable improvements with fine-tuned models, suggesting that alternative approaches may be needed to enhance the refinement process.

E. FOLIO vs LogicNLI

We observed that the effectiveness of our framework varied between the FOLIO and LogicNLI datasets. The semantic accuracy was generally higher for FOLIO, while syntactic accuracy was better for LogicNLI. We suspect that the higher semantic accuracy on FOLIO is due to its problems being more amenable to commonsense reasoning, allowing LLMs to leverage their pre-trained knowledge. In contrast, LogicNLI's design deliberately disentangles logical reasoning from commonsense inference, making it more challenging for LLMs to infer correct answers without strict logical deduction. The higher syntactic accuracy on LogicNLI likely stems from its simpler rule structure, with fewer logical relations and nesting levels compared to FOLIO's more complex rules.

F. Future Work

While our framework showed promising results in improving syntactic accuracy through grammar-constrained verification and dynamic example selection, there is still room for improvement in semantic accuracy. For future work, we would like to focus on enhancing the initial sketch generation process to better capture the meaning of logical problems, as well as testing this framework on tasks beyond logical reasoning, such as mathematical formula derivation, theorem proving and constraint satisfaction.

Additionally, we would like to test our framework on larger open-source refiners, such as the 70B of Llama 2 and LLama 3, as well as refiners with architectures different than Llama, such as Mistral [**?**] or Falcon [**?**].

Finally, it would be insightful to investigate the performance of grammar-constrained massive LLMs, like the GPT [**?**], Gemini [**?**] or some large variant of Llama [**?**]. While this technique is currently only available on open-source models, applying these techniques to state-of-the-art massive language models could potentially reduce the trade off between syntactic and semantic accuracy. This could encourage the development of new methods for constraining the output of black-box models or finding ways to efficiently implement grammar constraints in the decoding process of larger models.