Generating Domain Specific Natural Language SAT Reasoning Datasets

Sunandita Patra
JPMorgan AI Research, Chicago, USA

Keshav Ramani
JPMorgan AI Research, New York, USA

Daniel Borrajo
JPMorgan AI Research, Madrid, Spain

Sriram Gopalakrishnan
JPMorgan AI Research, New York, USA

Abstract

A key challenge for LLMs lies in their ability to reason. To evaluate an LLM's reasoning capabilities, there is a need for challenging natural language datasets for reasoning tasks. However, it is hard to manually generate these datasets across all domains, at the scale required by LLMs, as they require expensive effort from subject matter experts. As datasets become public, they become part of the training data of LLMs, leading to the need for newer datasets. In this work, we formalize the problem of synthetic generation of SAT reasoning-tasks of variable complexity in natural language that adhere to propositional logic. Then, we present our method, LTGen (Logical Text Generator), to generate custom datasets aligned with our formalism. We test `GPT-4o` and `o3-mini` on two auto-generated datasets using LTGen and find that LLMs struggle particularly on hard UNSAT problems and are biased toward predicting that the text is logically consistent.

1 Introduction

Many recent advances on Large Language Models (LLMs) focus on improving their ability to address reasoning tasks Mondorf and Plank [2024], Liu et al. [2025], Ballon et al. [2025]. However, complex logical reasoning still remains a challenging task for LLMs and Large Reasoning Models (LRMs), e.g., Wei et al. [2025], Hazra et al. [2025]. Here, we study their ability to handle a subset of complex reasoning tasks, specifically boolean satisfiability problems (SAT) when expressed in natural language with semantics related to a specific domain. Given that SAT formulation represents a standard framework into which many other complex reasoning tasks can be compiled into, the ability to reason about SAT would allow LLMs to reason on many other tasks.

A key aspect of evaluating a model's reasoning capabilities relates to the type of reasoning tasks that it can handle, and what limitations or biases are present. In Hazra et al. [2025], an analysis of thinking traces indicates that powerful reasoning models can internalize SAT-heuristic search procedures. Their work shows that even reasoning models do not necessarily have remarkable accuracy; the best model they tested achieves approximately 65% accuracy in generating a satisficing assignment for hard SAT tasks. Thus, it is key to understand the type of problems they can handle. The intuition is that a model's ability to solve these problems is closely tied to the computational complexity of the tasks themselves. This relationship serves as a central motivation for the investigations presented in this paper.

In order to train and evaluate models on reasoning tasks, we need datasets with the appropriate level of complexity and content. There is prior work on generating SAT reasoning tasks Wei et al. [2025]. However, this work is limited in how the text is generated; only logical-and, and logical-or operations are supported and the text is filled in with a rigid approach. In Qi et al. [2025], the authors also

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Efficient Reasoning.

generate reasoning tasks that can be configured by user input in a restricted way (by specifying a subject name and keyword). The objective in the generated tasks is to determine if a fact is true or false or uncertain. The SAT tasks are different in that they require reasoning to determine if a set of logic statements is satisfiable or not. This requires reasoning over the space of truth-value assignments for all propositions (the same problem as in Wei et al. [2025]); an example problem of such a task is determining if an HR policy in a company is consistent or not. See Section 5 for detailed related work.

We present a configurable generator of synthetic SAT reasoning tasks in natural language, LTGEN (Logical Text Generator). LTGEN's output can be used for evaluating the SAT reasoning capabilities of LLMs. One of the advantages over previous work relates to the wider variety of text generated (e.g., LTGEN supports conditional clauses in the text). LTGEN can also be used to generate SAT tasks for specific domains, and to filter generated samples based on the semantics and coherence of the text generated. This can be helpful in surfacing reasoning biases as shown in Dasgupta et al. [2022].

The main contributions of this paper are: (1) formalizing the problem of generating text data that adheres to propositional logic clauses; (2) defining a novel method, LTGEN, for creating domain-specific SAT datasets with coherent text using propositional logic, a SAT solver, and an LLM to systematically generate datasets tailored to specific domains; and, (3) demonstrating how the evaluation of reasoning performance of state-of-the-art LLMs can surface biases. We evaluate the performance of `GPT-4o` Hurst et al. [2024] and `o3-mini` OpenAI [2025] on LTGEN generated datasets, revealing that LLMs encounter difficulties especially with hard UNSAT problems and have a reasoning bias towards inferring logical consistency. Our findings underscore the need for improved methodologies in reasoning-task dataset generation and highlight the usefulness of datasets in evaluating LLM reasoning capabilities through domain-specific logic-based datasets.

2 Problem Description

We consider the problem of generating a custom dataset of textual descriptions of SAT tasks. Each element of the dataset is of the form $(C, \textit{text})$, where C is a set of propositional logic clauses and *text* is a natural language description adhering to C for a custom domain. The generated dataset, $\mathcal{D} = \left\{(C^{(i)}, \textit{text}^{(i)})\right\}_{i=1}^{N}$, provides a logically-aligned collection of clause set and text pairs, supporting the training and evaluation of language and reasoning models. We formulate the task in three stages: Abstract Clause Generation, Text Grounding, and Coherence Check.

Abstract Clause Generation: given a number of variables n_V, a number of clauses, n_C, and a set of logical operators O (e.g., $\{\neg, \vee, \wedge, \rightarrow\}$), generate a set of clauses $C = \{c_1, c_2, \ldots, c_{n_C}\}$ that involve n_V variables, $V = \{v_1, v_2, \ldots, v_{n_V}\}$, and n_C clauses. Each clause c_i should belong to $\mathcal{L}(V, O)$, which is the set of well-formed formulae over V using operators in O.

Text Grounding: given C (generated in the previous stage), generate $\textit{text} = \mathcal{T}(C, \mathcal{R}) \in Y$, where $\mathcal{T}$ is a stochastic grounding function such that *text* preserves the logical structure imposed by C. Y is the set of all possible grammatically correct sentences that can be derived from the reference text $\mathcal{R}$ from a given domain.

Coherence Check: given C and *text*, generated in the first and second stage, respectively, judge the logical alignment between C and *text*, and the quality and coherence of *text*.

3 Dataset Generation Architecture

In this section, we present our architecture, LTGEN, for generating a custom dataset (Figure 1). **Abstract Clause Generation**: LTGEN first generates a large number of unique abstract clauses by systematically following the rules to construct well-formed formulae, using a set of variables V of size n_V (also automatically generated), and operators, O. From this pool, LTGEN uniformly selects a subset of n_C clauses that are consistent (inconsistent) for SAT (UNSAT) items. For example, $C = \{v_1 \wedge \neg v_2\}$ belongs to the set of valid SAT formulae when $n_V = 2$ and $n_C = 1$. We use the Z3 solver (De Moura and Bjørner [2008]) to obtain ground-truth satisfiability labels (`consistent` vs `inconsistent`).

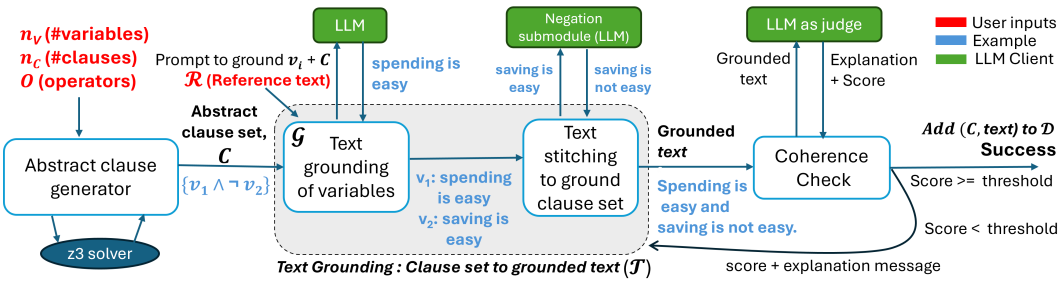


Figure 1: Overview of LTGEN, our dataset generation architecture.

Text Grounding: We use $\mathcal{R}$, a custom reference text from a given domain provided by the user, to *ground* each variable in V using an LLM. To formalize this stochastic mapping, we define a grounding function $\mathcal{G}$ that maps propositional variables to natural language segments. Given V, a clause set C, and a reference text $\mathcal{R}$, the grounding of a variable $v \in V$ is defined as $\mathcal{G}(v, C, \mathcal{R}) = v^{text}$, where v^{text} denotes a possible natural language realization of v. For $v_i, v_j \in V$, such that $i \neq j$, it must hold that $v_i^{text} \neq v_j^{text}$ ensuring unique text grounding of the variables. In this work, the grounding is done through prompting `o3-mini` with the arguments in the prompt (see Appendix A.3 for a full description of the prompts).

For a clause set C, we perform its text grounding, $\mathcal{T}(C, \mathcal{R})$, iteratively by processing one variable at a time using the variable grounding function, $\mathcal{G}(v_i, C, \mathcal{R})$. After grounding a variable, its grounded text replaces the symbolic variable in the clause set, and the result is included in the prompt for grounding the next variable. This sequential substitution ensures that the LLM generates assignments that are not only faithful to the logical structure but are also contextually coherent across variables. For example, in Figure 1, with,

$$C = \{v_1 \wedge \neg v_2\}, \text{ and } \mathcal{R} = \textit{financial savings},$$

a valid grounding is,

$$\mathcal{G}(v_1, C, \mathcal{R}) = \textit{spending is easy}, \text{ and } \mathcal{G}(v_2, C, \mathcal{R}) = \textit{saving is easy}.$$

If the LLM cannot generate any text related to $\mathcal{R}$ for a variable, the corresponding clause set is discarded from the dataset. Once LTGEN has generated valid text groundings of all the variables in C, the next step involves stitching the phrases together to form a coherent grounded text corresponding to C. LTGEN stitches the texts following a set of rules (see Appendix A.2). If the clause contains a negation, LTGEN uses an LLM to obtain the resulting text. In our example, the text grounding of $C = \{v_1 \wedge \neg v_2\}$ results in,

$$\mathcal{T}(C, \mathcal{R}) = \textit{Spending is easy and saving is not easy}.$$

Coherence Check: Finally, we use an LLM as a judge to provide explanations and scores (integer between 1 and 10) to judge the quality and coherence of the generated text, and use an iterative refinement loop to improve the variable text grounding.

The dataset generation procedure of LTGEN is detailed in Appendix A, including the pseudocode for its submodules.

Next, we present another example, Example 1 of generating a data item, $(C, \textit{text}) \in \mathcal{D}$.

Example 1. *We have the following parameter values:*

- *number of variables,* $n_V = 4$ *so that* $V = \{v_1, v_2, v_3, v_4\}$,
- *number of clauses,* $n_C = 4$,
- *Propositional Logic Operators,* $O = \{\wedge, \vee, \neg, \rightarrow\}$,
- *A relevant domain or application context,* $\mathcal{R}$ = *'banking'*,

Clause set: *A SAT clause set generated automatically using the Abstract Clause Set Generator is,* $C = \{(v_1 \to v_2), (v_2 \to v_3), (v_3 \vee \neg v_4)\}$.

Variable Text Grounding: LTGEN *grounds the variables, one at a time, using the grounding function* $\mathcal{G}$*. In this example,*

- $\mathcal{G}(v_1, C, \mathcal{R})$ = *The bank conducts regular compliance audits.*
- $\mathcal{G}(v_2, C, \mathcal{R})$ = *The bank maintains an integrated compliance management system.*
- $\mathcal{G}(v_3, C, \mathcal{R})$ = *The bank has established an internal fraud prevention unit.*
- $\mathcal{G}(v_4, C, \mathcal{R})$ = *The bank's compliance unit is adequately staffed.*

Grounded text for clause set: *The grounded text for the clause set,* C *is* $\mathcal{T}(C, \mathcal{R})$*, generated by stitching together the individual variable groundings,* $\bigcup_{i=1}^{4} \mathcal{G}(v_i, C, \mathcal{R})$*. In this example,* $\mathcal{T}(C, \mathcal{R})$ = *If the bank conducts regular compliance audits, then the bank maintains an integrated compliance management system. If the bank maintains an integrated compliance management system, then the bank has established an internal fraud prevention unit. The bank has established an internal fraud prevention unit or the bank's compliance unit is not adequately staffed.*

4 Experimental Evaluation

In this section, we present a preliminary evaluation of LLM performance on LTGEN generated datasets. We benchmark `GPT-4o` and `o3-mini` across three datasets, `hard3sat`, `uniform`, and `symbolic`, generated with $n_V \in [3, 19]$. The reference text, $\mathcal{R}$ is *banking* for `hard3sat` and banking derived themes for `uniform` (see Appendix B.4). `hard3sat` is a collection of hard 3-SAT problems with $n_C \approx 4.26 n_V$, i.e., problems in the phase-transition region of SAT (Mitchell et al. [1992]). `uniform` is a collection of uniformly generated clause sets (Appendix A.1). `symbolic` is a collection of 3-SAT tasks defined as clauses, not text, equivalent to only using the first phase of LTGEN. n_C vary from n_V to $5n_V$ in `uniform` and `symbolic`. In the coherence check module, we set the score threshold to be greater than or equal to 8 for an item to be included in the dataset. We report standard classification metrics to assess their ability to reason over logical consistency queries. `GPT-4o` and `o3-mini` are run with a temperature of 0, and max output-tokens as 4096 and 32768 respectively. All experiments were run on Intel Xeon Gold 6240R CPUs with 8 processors and 64GB RAM.

4.1 Results

Table 1 summarizes the performance of each LLM in determining logical consistency (SAT) when provided with a textually grounded (in the case of `hard3sat` and `uniform`) or symbolic clauses (`symbolic`). `o3-mini` consistently outperforms `GPT-4o` across all datasets and metrics, with the largest margin on the `uniform` dataset, indicating stronger performance in symbolic reasoning tasks. Both models exhibit higher recall than precision on `hard3sat` and `symbolic` datasets, suggesting a tendency to over-predict the `consistent` class. This bias is less pronounced on the uniform dataset, where precision and recall are better balanced. This suggests that the models seem biased toward predicting satisfiability when tasks are challenging. The number of valid samples (n) remains stable across runs, with occasional timeouts, especially for `o3-mini`, resulting in non-zero standard deviation in some rows. Analysis of `o3-mini`'s reasoning traces shows a methodical approach, consistent with prior observations (Ballon et al. [2025]). Across all datasets, `o3-mini` achieves near-perfect recall, indicating it almost never misclassified `consistent` as `inconsistent`. However, its lower precision shows that it often misclassified `inconsistent` as `consistent`.

4.2 Correlation of false positive rate with inconsistency

Table 2 reports the Pearson correlation coefficient (r) and p-value between the average LLM false positive rate (FPR) and the percentage of `inconsistent` ground truth samples for each dataset and LLM.

[1]Time measured on 45 samples

Table 1: Results for three datasets on `GPT-4o` and `o3-mini` over 5 runs per item. For each dataset and LLM we report the mean accuracy, precision, recall and F1 score (std dev in parentheses). 'time' column reports average LLM time. Values in bold represent the best performance for each dataset. n is the number of samples evaluated for each LLM.

Dataset	llm	accuracy	precision	recall	f1	n	time (s)
`hard3sat`	`GPT-4o`	0.44 (0.02)	0.36 (0.02)	0.81 (0.05)	0.49 (0.02)	128.0 (0.0)	5.14 (0.09)
`hard3sat`	`o3-mini`	**0.52 (0.02)**	**0.44 (0.02)**	**1.00 (0.00)**	**0.61 (0.02)**	111.8 (5.1)	70.18 (7.71)
`uniform`	`GPT-4o`	0.62 (0.02)	0.65 (0.03)	0.44 (0.04)	0.52 (0.04)	158.0 (0.0)	4.79 (0.07)
`uniform`	`o3-mini`	**0.83 (0.01)**	**0.76 (0.01)**	**0.96 (0.02)**	**0.85 (0.01)**	158.0 (0.0)	18.78 (0.74)
`symbolic`	`GPT-4o`	0.51 (0.04)	0.50 (0.03)	0.88 (0.05)	0.64 (0.03)	168.4 (1.5)	6.70 (0.42) [1]
`symbolic`	`o3-mini`	**0.67 (0.01)**	**0.62 (0.01)**	**0.99 (0.01)**	**0.76 (0.01)**	157.8 (2.6)	38.03 (5.10)[1]

Table 2: Correlation between LLM false positive rate (FPR) and the percentage of `inconsistent` inputs, grouped by the number of clauses. n_{groups} indicates the total number of input groups formed based on clause count. r represents Pearson correlation, and p-value denotes statistical significance.

Dataset llm	`hard3sat` `GPT-4o`	`hard3sat` `o3-mini`	`symbolic` `GPT-4o`	`symbolic` `o3-mini`	`uniform` `GPT-4o`	`uniform` `o3-mini`
r	0.94	0.88	0.97	0.91	0.39	0.41
p-value	<0.001	<0.001	<0.001	<0.001	0.003	0.002
n_{groups}	41	41	46	46	56	56

If the classifier is unbiased, i.e., its predictions do not depend on the underlying class proportions, then the false positive rate (FPR) should remain stable as the proportion of `inconsistent` problems changes. However, for the `symbolic` and `hard3sat` datasets, both models show strong positive correlations ($r > 0.75$) between FPR and the proportion of `inconsistent` problems. This indicates that as the `inconsistent` class becomes more prevalent, the models make more false positive errors, revealing a systematic bias: LLMs are more likely to incorrectly predict `consistent` when the ground truth is `inconsistent`. On the `uniform` dataset, these correlations are weaker, suggesting that the relationship between FPR and ground truth inconsistency is less pronounced. The statistical significance (p-values) confirms the strength of these correlations. The discussion in Section 4.3 further illustrates this tendency, showing that LLMs systematically overpredict `consistent` across the `symbolic` and `hard3sat` datasets, regardless of whether the input is textual or symbolic.

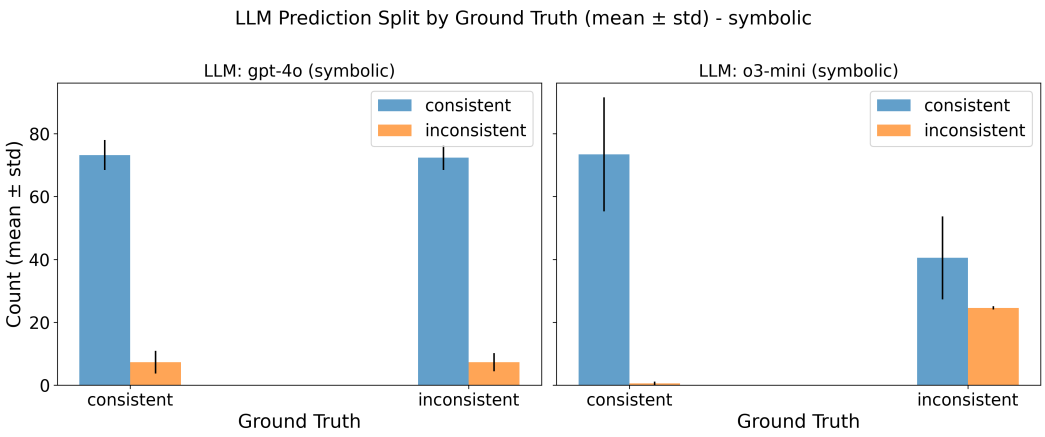
4.3 Prediction split by ground truth for each LLM

Figure 2 illustrates the distribution of LLM predictions for each ground truth class, separated by model. Each subplot corresponds to a different LLM, and the bars show the percentage of outputs assigned to each prediction category (`consistent` and `inconsistent`) for both ground truth labels. A systematic bias to predict inputs as `consistent` is observed across the `symbolic` and `hard3sat` datasets. Irrespective of the LLM or dataset, `consistent` is predicted far more frequently than `inconsistent`. This bias, however, is not observed for the `uniform` dataset.

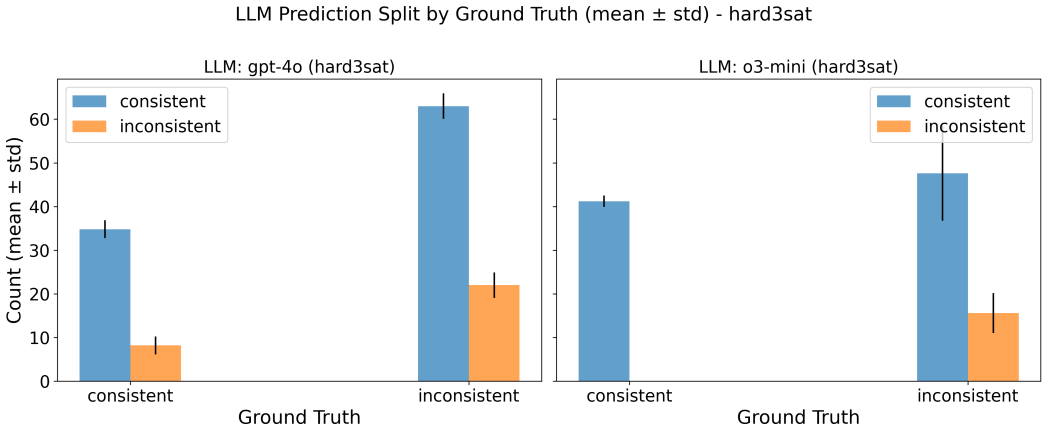
5 Related Work

Current empirical results in the literature indicate that LLMs and LRMs cannot consistently solve all problems in the boolean satisfiability setting even for a restricted number of variables and clauses Hazra et al. [2025]. However, by adding additional computation effort through thinking tokens, reasoning models are able to solve many reasoning problems Ballon et al. [2025]. This can work well in many cases if the task in the prompt is presented in a format that makes it easy to identify the propositions and the structure of the clauses as seen in the results of existing work Liu et al. [2025], Ballon et al. [2025].

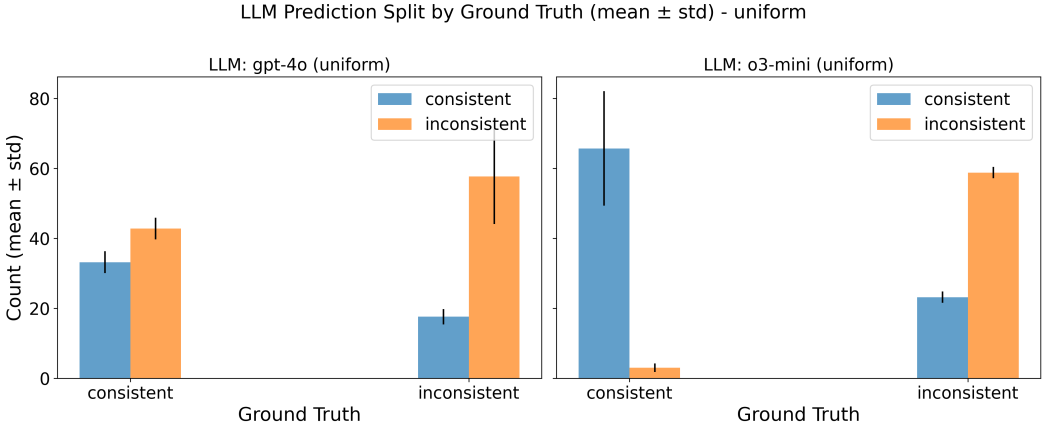
However, if the problem in the prompt is represented as natural language that is not easily translated to propositions and clauses, then the performance may differ, especially if the vocabulary or structure



(a) `symbolic` Dataset



(b) `hard3sat` Dataset



(c) `uniform` Dataset

Figure 2: Prediction split for each LLM when compared with the ground truth.

of the language is very domain specific and unseen in a model's training data. There is existing work on dataset generation for this problem that can be seen as natural language translation of logic problems Wei et al. [2025], Morishita et al. [2024], Qi et al. [2025]. Such works adopt a similar approach to ours in that the reasoning problems are first generated in a canonical form and then the propositions or variables are replaced with text. However, the text in the problems generated is often rigidly generated; in Wei et al. [2025] objects and properties are chosen or generated independently and replace propositions without considering the entire text semantics, and thus text coherence can be bad. The rigid structure of the text also makes it easy to translate back to a formal logic representation to do reasoning or use a tool. This may not be the case in all text, and our work supports a more general reasoning problem generator where the text is more coherent to a topic or theme and supports more logical connectives. In Qi et al. [2025] the problems generated only ask if a goal statement is true, false, or uncertain. This is different from the tasks we generate, which requires a model to reason if a set of statements is satisfiable or not; i.e., reasoning over the space of all possible truth assignments. Their text grounding procedure is also more rigid, in that predicate phrases are limited to a maximum of five words.

A different method described in Morishita et al. [2024] generates a reasoning trace designed explicitly to train LLMs to do reasoning, and some of the samples have (intentionally) wrong steps. The procedure used to generate natural language text is syntactically well formed, but it does not include semantics. In our work, we try to add semantic content from a particular domain. This would also make the synthetic samples of a domain more realistic for evaluation. It can also help surface any biases in reasoning on that domain as existing research has shown that LLMs can be biased toward stating logical consistency for results that agree with their training data even if the argument was incorrect and vice versa Dasgupta et al. [2022].

In our dataset generator, we can vary the number of propositions, number of clauses, and connectives used in the clauses which are not variable in Morishita et al. [2024]. Some of our problem-generation parameters are found in Wei et al. [2025], albeit they are limited to Conjunctive Normal Form (CNF) formulae. CNF is very restrictive in terms of the clause complexity and how expressive the natural language can be. While any satisfiability task can be compiled into CNF, real natural language problems can include conditional statements (implications) in the text, which are not used in Wei et al. [2025]. Our approach also enables easy generalization to other domains, and we evaluate the semantic coherence (i.e. is the text meaningful or not) with LLM-as-a-judge to improve sample quality.

6 Limitations and Conclusion

We introduced the LTGEN framework to generate custom SAT reasoning datasets by aligning propositional logic with domain-specific language through a clause generation and text grounding procedure. Datasets built with LTGEN can help evaluate the limits of various reasoning models on domain specific content that they would be used for. Our experiments showed that LTGEN can generate challenging SAT reasoning problems, and can uncover inherent biases; one such bias is a tendency to favor satisfiability in complex UNSAT cases for models such as `GPT-4o` and `o3-mini`. In future work on LTGEN, we plan to further improve the text generation process, and address dataset imbalances, especially when the number of variables and clauses are large.

References

Marthe Ballon, Andres Algaba, and Vincent Ginis. The relationship between reasoning and performance in large language models–o3 (mini) thinks harder, not longer. *arXiv preprint*

arXiv:2502.15631, 2025.

Ishita Dasgupta, Andrew K Lampinen, Stephanie CY Chan, Antonia Creswell, Dharshan Kumaran, James L McClelland, and Felix Hill. Language models show human-like content effects on reasoning. *arXiv preprint arXiv:2207.07051*, 2(3), 2022.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

Rishi Hazra, Gabriele Venturato, Pedro Zuidberg Dos Martires, and Luc De Raedt. Have large language models learned to reason? a characterization via 3-sat phase transition. *arXiv preprint arXiv:2504.03930*, 2025.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Hanmeng Liu, Zhizhang Fu, Mengru Ding, Ruoxi Ning, Chaoli Zhang, Xiaozhang Liu, and Yue Zhang. Logical reasoning in large language models: A survey, 2025. URL `https://arxiv.org/abs/2502.09100`.

David Mitchell, Bart Selman, Hector Levesque, et al. Hard and easy distributions of sat problems. In *AAAI*, volume 92, pages 459–465, 1992.

Philipp Mondorf and Barbara Plank. Beyond accuracy: Evaluating the reasoning behavior of large language models – a survey, 2024. URL `https://arxiv.org/abs/2404.01869`.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. Enhancing reasoning capabilities of llms via principled synthetic logic corpus. *Advances in Neural Information Processing Systems*, 37:73572–73604, 2024.

OpenAI. Openai o3-mini system card. `https://cdn.openai.com/o3-mini-system-card-feb10.pdf`, 2025. Accessed: 2025-08-29.

Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. Large language models meet symbolic provers for logical reasoning evaluation. *arXiv preprint arXiv:2502.06563*, 2025.

Anjiang Wei, Yuheng Wu, Yingjia Wan, Tarun Suresh, Huanmi Tan, Zhanke Zhou, Sanmi Koyejo, Ke Wang, and Alex Aiken. Satbench: Benchmarking llms' logical reasoning via automated puzzle generation from sat formulas. *arXiv preprint arXiv:2505.14615*, 2025.

A Methodology Details

In this section, we describe LTGEN's dataset generation procedure (Algorithm 4) in depth, including the details and pseudocode of its submodules, Abstract Clause Generation (Section A.1) and Text Grounding (Section A.2). We include the prompts provided to the LLMs in Section A.3.

A.1 Abstract Clause Generation

LTGEN's abstract clause generation procedure (Algorithm 1) is designed to create a set of clause sets, where each clause set, C, contains a fixed number of propositional logic clauses. The algorithm has the following input parameters:

- number of symbolic variables n_V (with $V = \{v_1, v_2, \ldots, v_{n_V}\}$),
- the number of clauses per clause set, n_C,
- the desired number of unique clause sets N (the required size of the dataset, $\mathcal{D}$),
- a generation mode (which can be either `uniform` or `3sat`),

- a maximum depth parameter (max_depth) that controls the complexity of the clause structures, and,
- a target satisfiability status ($target\_sat\_status$) indicating whether the generated clause sets should be SAT or UNSAT.

In the `uniform` mode, the algorithm first generates a collection of abstract clause structures by using a procedure, `GenerateClauseStructures`. This is to increase the diversity of clause structures in the datasets. `GenerateClauseStructures` generates clause structures with a placeholder 'var' where variables can be positioned. These structures act as templates for the clauses. Next, LTGEN substitutes the placeholder 'var's within the clause structures with symbolic variables from V using the function `SubstituteVariables`, thereby creating a unique set of well-formed formulae, *clause_candidates*. LTGEN samples subsets of size n_C from *clause_candidates* until N valid clause sets are collected to form the set $\mathcal{S}$. In each iteration, a candidate clause set C' of size n_C is selected and checked for logical consistency. If C' is consistent (inconsistent), `target_sat_status` is SAT (UNSAT) and C' is not already part of the $\mathcal{S}$, C' is added to $\mathcal{S}$.

Alternatively, if the mode is set to `3sat`, the algorithm doesn't preconstruct the clause structures like in `uniform` mode, since 3-CNF form has a fixed structure (e.g., $(v_1 \vee \neg v_2 \vee v_3) \wedge (\neg v_1 \vee v_2 \vee \neg v_3) \wedge ....$) It directly generates clause sets in 3-CNF form using the `Generate3SATInstance` procedure with the variable set V and the number of clauses n_C. As with the `uniform` mode, a candidate C' undergoes a consistency check, and if its satisfiability status aligns with the desired `target_sat_status`, it is included in the dataset.

Once LTGEN has N valid clause sets, the algorithm outputs the collection $\mathcal{S} = \{C^{(1)}, C^{(2)}, \ldots, C^{(N)}\}$. This method offers a flexible and controlled approach for generating abstract clause sets for our dataset.

Algorithm 1: Abstract Clause Generation

Input: n_V, n_C, O, N, mode, max_depth, $target\_sat\_status$
Output: A set of clause sets $\mathcal{S} = \{C^{(i)}\}_{i=1}^{N}$

$V \leftarrow \{v_1, v_2, ..., v_{n_V}\}$ # a set of symbolic variables
$\mathcal{S} \leftarrow \emptyset$ # Set of clause sets to include in the dataset, $\mathcal{D}$
if *mode = uniform* **then**
  clause_structures $\leftarrow$ **GenerateClauseStructures**($'var', max\_depth, n_C, O$)
  clause_candidates $\leftarrow$ **SubstituteVariables**(V, *clause_structures*)
  while $|\mathcal{S}| < N$ **do**
    $C' \leftarrow$ Subset of *clause_candidates* of size n_C
    if *IsConsistent(C′)* $\wedge$ *(target_sat_status = SAT)* $\wedge$ $(C' \notin \mathcal{S})$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup C'$
    else if *(Not IsConsistent(C′))* $\wedge$ *(target_sat_status = UNSAT)* $\wedge$ $(C' \notin \mathcal{S})$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup C'$
else if *mode = 3sat* **then**
  while $|\mathcal{S}| < N$ **do**
    $C' \leftarrow$ **Generate3SATInstance**(V, n_C)
    if *IsConsistent(C′)* $\wedge$ *(target_sat_status = SAT)* $\wedge$ $(C' \notin \mathcal{S})$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup C'$
    else if *(Not IsConsistent(C′))* $\wedge$ *(target_sat_status = UNSAT)* $\wedge$ $(C' \notin \mathcal{S})$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup C'$
return $\mathcal{S}$

A.2 Text Grounding: Clause Set to Grounded Text

Our text grounding procedure, Algorithm 2, is aimed at converting a set of symbolic clauses into a coherent natural language representation. LTGEN starts by mapping each symbolic variable v_i in the set V to a grounded text using an LLM, $\mathcal{M}$. The prompt depends on the reference text provided by the user ($\mathcal{R}$), and, the chosen grounding technique g, which may be one of 'broadtopic', 'themes', or 'fulltext'. In 'broadtopic', each variable is grounded using only a high level topic (e.g., a domain name

or subject area), for example, 'banking'. In 'themes', the grounding is based on a set of intermediate themes or concepts extracted from the domain. Please see Appendix B.4 for some examples of themes. Finally, in the 'fulltext' setting, an entire paragraph of reference text can be used directly to guide the grounding. We used `o3-mini` as $\mathcal{M}$ in our experiments.

Upon receiving a valid response from $\mathcal{M}$, the algorithm directly substitutes the symbolic variable with the grounded text within the clause. This iterative replacement process continues for every variable in V, and the updated text is provided as a part of the prompt for grounding the next variable. Should $\mathcal{M}$ fail to produce a valid response for any variable, the procedure terminates, indicating a failure in the grounding process. Once LTGEN has grounded all the variables in V successfully, LTGEN procedurally stitches them together, following a set of rules specific to the operators present in the clause, as outlined in Algorithm 3.

Algorithm 2: Text Grounding : Clause Set to Grounded Text

Input: V: set of variables, C: set of clauses, $\mathcal{R}$: reference text for grounding
$\mathcal{M}$: language model # we use `o3-mini`
g: grounding technique # LTGEN supports `broadtopic`, `themes`, and, `fulltext`
Output: *text* # text grounding of C

text $\leftarrow$ `str`(C) # initialize with the string representation of the symbolic clauses
$var\_to\_text \leftarrow \{\}$ # dictionary mapping each variable to text
foreach $v_i \in V$ **do**
    $prompt \leftarrow$ `VARIABLE_TO_TEXT_GROUNDING_PROMPT` + $\mathcal{R}$ + *text* (See section A.3 for a full description of the prompt)
    $response \leftarrow \mathcal{M}(prompt)$
    if $response \neq failure$ **then**
        $var\_to\_text[v_i] \leftarrow response$
        text $\leftarrow$ Replace v_i with $var\_to\_text[v_i]$
    else
        return failure
Text Stitching: Generate coherent text using the individual variable groundings
text $\leftarrow$ Do_Text_Stitching$(C,\ var_to_text, \mathcal{M})$
return *text*

One of the challenges we faced was duplicate assignment of text to the variables during the grounding procedure. Some prompt tuning and grounding the variables one at a time solved the issue. We tried to do the stitching step using LLMs, however, we faced some hallucination by the LLM. As a result, we came up with a deterministic procedure to do the stitching, Algorithm 3, which works reasonably well for clauses with nested depth up to 3.

Finally, Algorithm 4 outlines LTGEN's pseudocode of dataset generation using Algorithms 1 and 2 as submodules.

Algorithm 3: Text Stitching: Clause Set to Coherent Grounded Text

Input: C: set of clauses
var_to_text: dictionary mapping variables to grounded text
$\mathcal{M}$: language model # we use `o3-mini`
Output: *text* # coherent text from C

```
# Main Routine: Do_Text_Stitching
text ← empty string
foreach clause ∈ C do
    stitched_text ← Convert_Single_Clause_To_Text(clause, var_to_text, M)
    text ← stitched_text + text
return text

# Subroutine: Convert_Single_Clause_To_Text
Function Convert_Single_Clause_To_Text(clause, var_to_text, M):
    if clause ∈ var_to_text # clause is a single variable then
        return var_to_text[clause]
    sub_text_list ← []
    foreach sub_clause ∈ clause do
        sub_text ← Convert_Single_Clause_To_Text(sub_clause, var_to_text, M)
        sub_text_list.append(sub_text)
    if is_and(clause) then
        return Connect_Sentences(sub_text_list, AND)
    else if is_or(clause) then
        return Connect_Sentences(sub_text_list, OR)
    else if is_not(clause) then
        negation_prompt ← NEGATION_PROMPT + sub_text_list[0]
        llm_response ← M(negation_prompt)
        if llm_response is a valid negation then
            return llm_response
        else
            return "Negation Failure"
    else if is_implies(clause) then
        return Connect_Sentences(sub_text_list, IMPLIES)

# Subroutine: Connect_Sentences
Function Connect_Sentences(S, connector):
    n ← |S|
    if n = 2 then
        s1, s2 ← S[0], S[1]
        if connector = AND then
            combined_sentence ← "Both, s1, and, s2."
        else if connector = OR then
            combined_sentence ← "Either s1, or, s2."
        else if connector = IMPLIES then
            combined_sentence ← "If s1, then, s2."
    else if n = 3 then
        s1, s2, s3 ← S[0], S[1], S[2]
        if connector = AND then
            combined_sentence ← "s1, s2, and s3."
        else if connector = OR then
            combined_sentence ← "Either s1, or, s2, or s3."
    return combined_sentence
```

Algorithm 4: LTGEN (Generate Dataset $\mathcal{D}$)

Input:
For Clause Generation:
n_V, n_C, O, N
$mode$, max_depth, $target\_sat\_status$
For Text Grounding:
$\mathcal{R}$: reference text for grounding,
$\mathcal{M}$: language model (e.g., `o3-mini`),
g: grounding technique (LTGEN supports `broadtopic`, `themes`, or `fulltext`)
Output: Dataset $\mathcal{D} = \{(C^{(i)}, \textit{text}^{(i)})\}_{i=1}^{N}$

$V \leftarrow \{v_1, v_2, ..., v_{n_V}\}$ # a set of symbolic variables
$\mathcal{S} \leftarrow$ **AbstractClauseGeneration**$(n_V, n_C, O, N, mode, max\_depth, target\_sat\_status)$
$\mathcal{D} \leftarrow \emptyset$
foreach $C \in \mathcal{S}$ **do**
    # Try up to K times to ground and validate, we used $K = 5$
    accepted $\leftarrow$ false
    for $k \leftarrow 1$ **to** K **do**
        text $\leftarrow$ **TextGrounding**$(V, C, \mathcal{R}, \mathcal{M}, g)$
        if *text* $=$ *failure* **then**
            continue
        # Score the grounded text with the model
        $explanation, s \leftarrow \mathcal{M}(\texttt{COHERENCE\_CHECK\_PROMPT} + C + \textit{text})$ # See A.3 for the full prompt
        if $s \geq$ *threshold* **then**
            $\mathcal{D} \leftarrow \mathcal{D} \cup \{(C, \textit{text})\}$
            accepted $\leftarrow$ true
            break
return $\mathcal{D}$

A.3 Prompts to the LLM

In this section, we present the prompts that LTGEN sends to the LLM clients for variable text grounding, negating text, and evaluating coherence. We also present the prompt used for the experimental evaluation of our datasets with `GPT-4o` and `o3-mini`.

```
VARIABLE_TO_TEXT_GROUNDING_PROMPT =

"Your task is to provide text assignments for variables following the
instructions below.

I will provide you with a list of variables and clauses, and reference
text.

Please assign simple statements/phrases derived from the text to each
variable such that:

- All the clauses should evaluate to true.

- The clauses conform to the information in the text.

Use the reference text as guidance to fill in the variable <var> with
simple, realistic phrases or sentences.

Given a set of clauses (connected with propositional logic operators: And,
Or and Not) ensure each clause remains consistent and meaningful when the
variables are replaced with your suggestions.

Variables should have different text assignments without overlap.
```

```
 The text assignments should be atomic.  Do not include logical connectors
like and, or, not in the text groundings.

 - Clarity:  Keep the text grounding as simple as possible, ensuring atomic
sentences or phrases.

 - Consistent:  Ensure the grounded text aligns with the overall theme and
is related to real-world contexts.

 - Theme:  The overall theme should be realistic.

 Output the variable-to-text mapping as a JSON object with variable names
as key and its corresponding assigned text as value. "

 if creating SAT data items:  attach CONSISTENT_EXAMPLES to the prompt

 else:  attach INCONSISTENT_EXAMPLES to the prompt

 CONSISTENT_EXAMPLES = (

 NOT clause examples:

 Suppose a clause is Not(v1) and the topic is driving.

 One possible variable text assignment out of many for v1 is "Sam can
drive."

 The clause text grounding then becomes "Sam cannot drive."

 Suppose a clause is Not(v2) and the topic is travelling.

 One possible variable text assignment out of many for v2 is "Sam has
travelled the world."

 The clause text grounding then becomes "Sam has not travelled the world."

 AND clause examples:

 Suppose a clause is And(v1, v2) and the topic is travelling.

 One possible variable text assignment out of many for v1 is "Sam has
visited Europe."

 One possible variable text assignment out of many for v2 is "Sam has
visited Australia."

 The clause text grounding then becomes:  "Sam has visited Europe and Sam
has visited Australia."

 Suppose a clause is And(v3, v4) and the topic is weather.

 One possible variable text assignment out of many for v3 is "The weather
is sunny."

 One possible variable text assignment out of many for v4 is "The beach is
beautiful on sunny days."

 The clause text grounding then becomes:  "The weather is sunny and the
beach is beautiful on sunny days."

 OR clause examples:

 Suppose a clause is Or(v1, v2) and the topic is driving.

 One possible variable text assignment out of many for v1 is "Sam can drive
a car."

 One possible variable text assignment out of many for v2 is "Sam can drive
a bike."

 The clause text grounding then becomes:  "Sam can drive a car or Sam can
drive a bike."
```

Nested formulas:

Nested NOT and AND clause examples:

Suppose a clause is Not(And(v1, v2)) and the topic is driving.

One possible variable text assignment out of many for v1 is "Sam can drive a car."

One possible variable text assignment out of many for v2 is "Sam can drive a bike."

The clause text grounding then becomes "Sam cannot drive a car or Sam cannot drive a bike."

Nested AND and OR clause examples:

Suppose a clause is And(v1, Or(v2, v3)) and the topic is travelling.

One possible variable text assignment out of many for v1 is "Sam has visited Europe."

One possible variable text assignment out of many for v2 is "Sam has visited Australia."

One possible variable text assignment out of many for v3 is "Sam has visited UK."

The clause text grounding then becomes: "Sam has visited Europe. Sam has visited Australia or Sam has visited UK."

Nested OR and NOT clause examples:

Suppose a clause is Or(Not(v1), v2) and the topic is weather.

One possible variable text assignment out of many for v1 is "The weather is sunny."

One possible variable text assignment out of many for v2 is "The beach is open."

The clause text grounding then becomes "The weather is not sunny or the beach is open."

)

INCONSISTENT_EXAMPLES = (

It is alright if the whole text is logically inconsistent here.

NOT clause examples:

Suppose clauses are v1 AND Not(v1).

One possible variable text assignment out of many for v1 is "Sam can drive."

The clause text grounding then becomes "Sam can drive and Sam cannot drive."

(But suppose another clause says "Sam cannot drive.")

Suppose a clause is Not(v2) AND And(v1, v2).

One possible variable text assignment out of many for v1 is "Sam can drive."

One possible variable text assignment out of many for v2 is "Sam has travelled the world."

The clause text grounding then becomes "Sam has not travelled the world. Sam can drive and Sam has travelled the world."

(But suppose another clause says "Sam has not travelled the world.")

AND clause examples:

Suppose a clause is And(v1, v2) AND And(Not(v1), Not(v2)).

One possible variable text assignment out of many for v1 is "Sam can drive."

One possible variable text assignment out of many for v2 is "Sam has travelled the world."

The clause text grounding then becomes: "Sam can drive and Sam has travelled the world. Sam cannot drive and Sam has not travelled the world."

Suppose a clause is And(v3, v4) AND And(Not(v3), v4).

One possible variable text assignment out of many for v3 is "The weather is sunny."

One possible variable text assignment out of many for v4 is "The park is open."

The clause text grounding then becomes: "The weather is sunny and The park is open. The weather is not sunny and The park is open."

OR clause examples:

Suppose a clause is Or(v1, v2) AND And(v1, Not(v2)).

One possible variable text assignment out of many for v1 is "Sam can drive."

One possible variable text assignment out of many for v2 is "Sam has travelled the world."

The clause text grounding then becomes "Sam can drive or Sam has travelled the world. Sam cannot drive and Sam has not travelled the world."

(But suppose another clause says "Sam cannot drive and Sam has not travelled the world.")

)

COHERENCE_CHECK_PROMPT:

"I want you to evaluate the following text in terms of coherence and consistency. Is the text consistent and factually correct according to the reference text? Please provide a score between 1 and 10 with your explanation. Put the score inside <score> and </score>. The clauses corresponding to the text are: " + str($\mathcal{C}$) + "The text is *text*"

NEGATION_PROMPT:

"Only provide the answer in plain English inside <negation> and </negation>. What is the negation of the following text? The text is " + *text*

EXPERIMENTAL_EVALUATION_PROMPT:

"Is the following set of statements logically consistent or inconsistent?

Explain briefly and answer yes/no. Put your final answer(consistent/inconsistent) inside <result> and </result>. Statements: *text*

B Additional Experimental Results

Subsections B.1 and B.3 study the change in accuracy by using different proxies for input complexity such as number of variables, and length of input. Subsection B.2 studies the stability of outputs across several runs against an increase in number of variables, and 4.3 studies the performance of the LLMs across the ground truth classes for both datasets.

B.1 Accuracy as a function of number of variables

Figure 3 shows the accuracy of each LLM as a function of the number of variables in the logical consistency queries. Each line represents a different LLM, and the x-axis indicates the number of variables present in the problem instance. The plot reveals how model performance varies with increasing problem complexity – where the number of variables can be used a proxy to represent problem complexity. Higher accuracy at larger variable counts suggests better generalization to more complex symbolic reasoning tasks.

We notice that across all the three datasets, there is a reduction in accuracy with an increase in the number of variables. As a general trend, we notice that `o3-mini` performs better than `GPT-4o`. This can be attributed to `o3-mini`'s superior performance in reasoning tasks.

B.2 Homogeneity of LLM outputs vs num variables

Figure 4 presents the homogeneity of LLM outputs as a function of the number of variables in the logical consistency queries. We define Homogeneity as follows:

Let S be the set of outputs produced by an LLM for a given query across multiple runs, and let v_i denote the i-th unique output value in S. The homogeneity H for a query is defined as the percentage of outputs that correspond to the most frequently occurring value:

$$H = \frac{\max_{v_i}(\text{count}(v_i))}{|S|} \times 100$$

where count(v_i) is the number of times value v_i appears in S, and $|S|$ is the total number of outputs for the query. Higher values of H indicate greater consistency in the LLM's responses across repeated runs for the same query.

To analyze how output consistency varies with problem size, we compute the average homogeneity $\overline{H}_n$ for each value of the number of variables n:

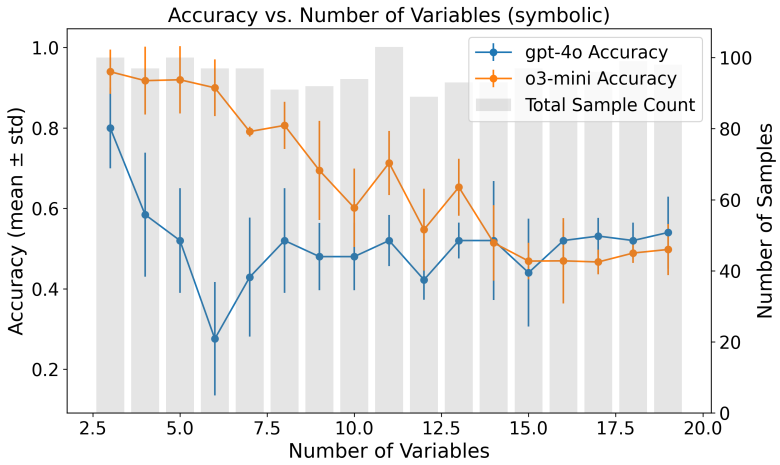
$$\overline{H}_n = \frac{1}{|Q_n|} \sum_{q \in Q_n} H(q)$$

where Q_n is the set of queries with n variables, and $H(q)$ is the homogeneity for query q as defined above. This metric summarizes the typical output stability of an LLM for all queries of a given number of variables, enabling direct comparison of models as logical complexity increases. Higher $\overline{H}_n$ values indicate that the LLM produces more repeatable outputs for larger or more complex queries.

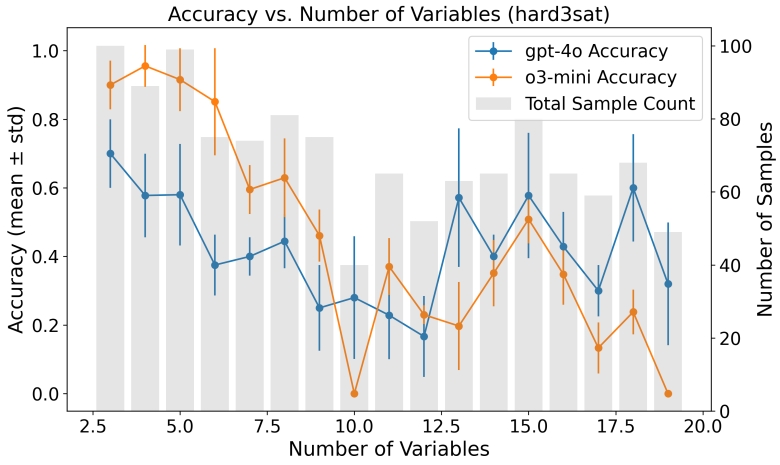
We notice that for the `symbolic` dataset, the LLMs tend to produce homogenous outputs by and large for most queries of a given number of variables. `o3-mini` in particular is more stable in its homogeneity, and achieves mostly values between 90% to 100%. In the case of the `hard3sat` dataset, we notice a difference in performance as `o3-mini` tends to maintain its average homogeneity, whereas there is a clear degradation for `GPT-4o`. In the case of `uniform` however, there is a general reduction in homogeneity for both models.

B.3 Input vs Output Length Analysis

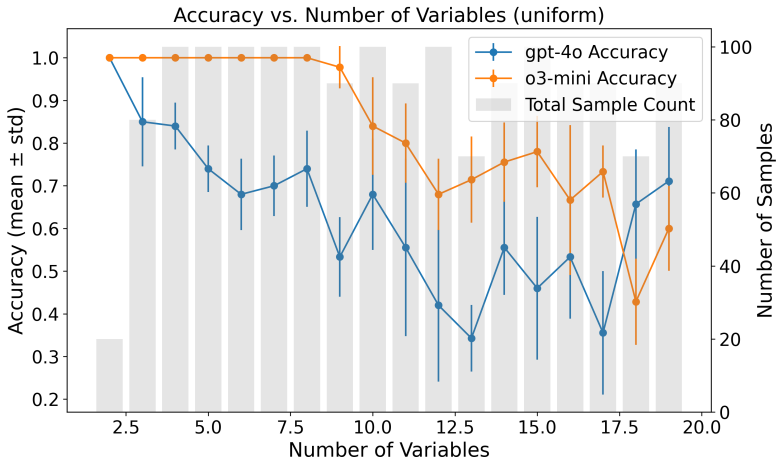
Figure 5 visualizes the relationship between the length of the input (grounded text) and the output (LLM response) for each experiment, with points colored by LLM. The horizontal axis represents



(a) `symbolic` Dataset

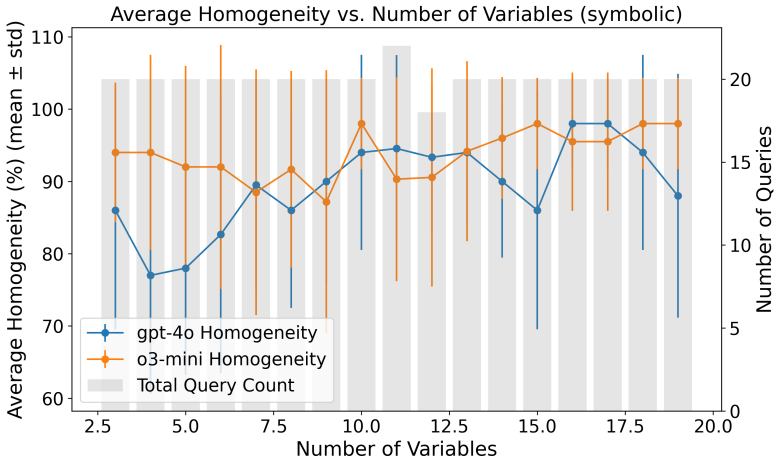


(b) `hard3sat` Dataset

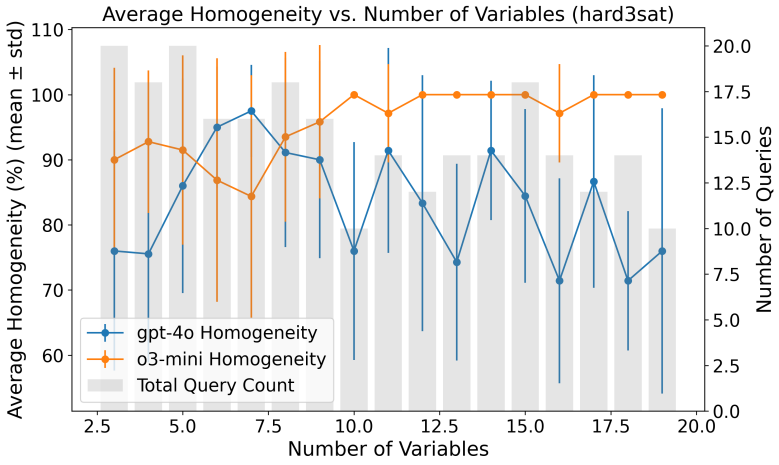


(c) `uniform` Dataset

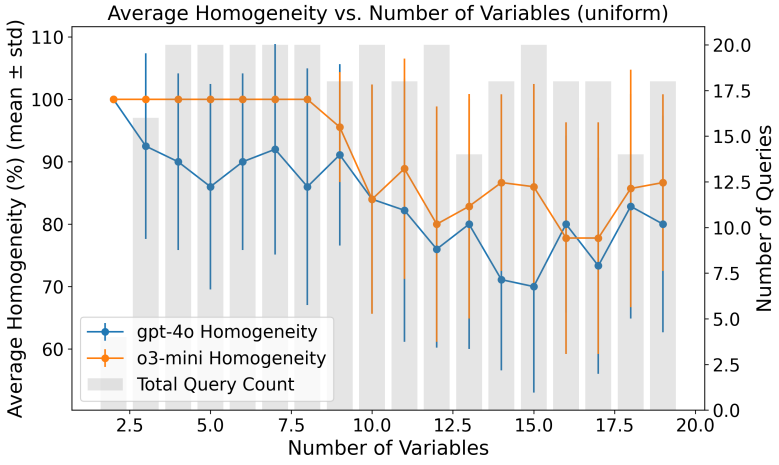
Figure 3: Accuracy as a Function of number of variables.



(a) `symbolic` Dataset



(b) `hard3sat` Dataset

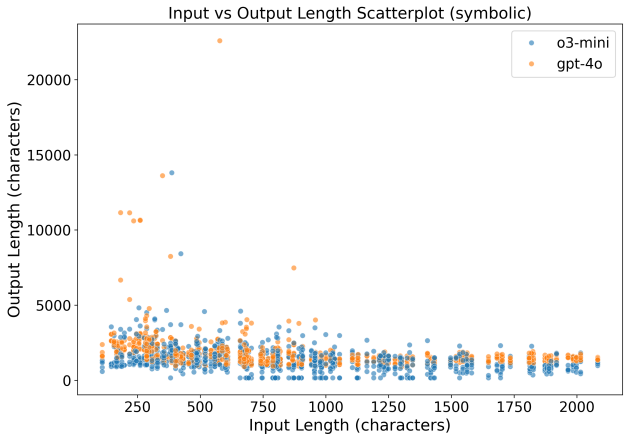


(c) `uniform` Dataset

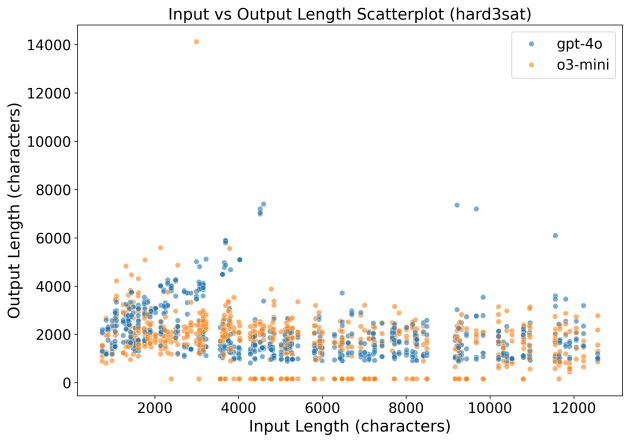
Figure 4: Homogeneity as a function of number of variables

the number of characters in the input, while the vertical axis shows the number of characters in the output. Each point corresponds to a single query-response pair.

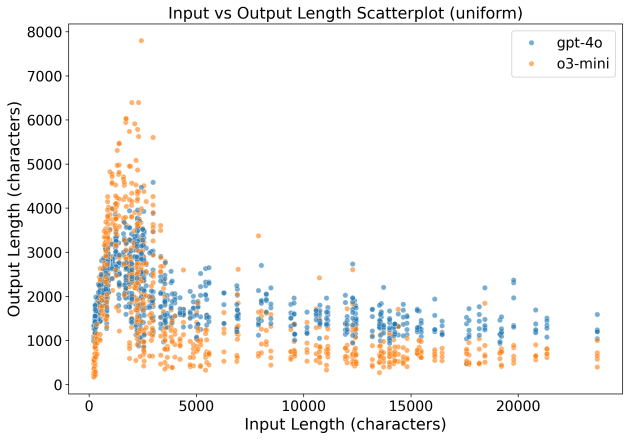
A linear trend in the scatterplot would indicate that the LLM output length scales proportionally with input length, while a dispersed pattern suggests variable verbosity or reasoning style. This plot helps to understand whether LLMs tend to over-generate or under-generate text as input size increases, and whether their output style is consistent across experiments.



(a) `symbolic` Dataset



(b) `hard3sat` Dataset



(c) `uniform` Dataset

Figure 5: Length of output (characters) vs Length of input (characters).

We notice that in the case of the `symbolic` dataset, the output of both LLMs tend to fairly be within 5000 characters. This could be attributed to the queries being of non verbal nature. In the case of `hard3sat` however, there is a steady linear increase in the outputs of `GPT-4o` until input length of roughly 5000 characters, after which outputs are at most 4000 characters. `o3-mini` for this dataset tends to produce outputs that are mostly under 5000 characters in length. In the `uniform` dataset however, there is interesting behavior demonstrated by both models. There is a sharp increase in output legnth for `GPT-4o`, reaching lengths of up to 5000 for inputs up to 4000 characters in length, after which output length falls off and remains contained between 1000 characters and 3000 characters. In the case of `o3-mini`, this trend is even more pronounced, withh output lengths reaching up to 7000 characters for inputs up to 3000 characters, after which there is a dramatic reduction in output length, being confined to within 3000 characters. These plots show the interesting behaviors exhibited by both LLMs for different datasets.

B.4 Dataset Distribution

Tables 3 and 4 shows the number of SAT and UNSAT problems in the `hard3sat`, `uniform` and `symbolic` datasets. `hard3sat` and `symbolic` are generated with $O = \{\wedge, \vee, \neg\}$, and `uniform` is generated with $O = \{\wedge, \vee, \neg, \rightarrow\}$. Tables 3 shows the problem counts grouped by the number of variables. Tables 4 shows the problem counts grouped by various clause intervals. As shown in Table 5, both the grounded text length and the number of sentences generally increase with the number of variables. Similarly, Table 6 demonstrates a consistent growth in text length and sentence count as the number of clauses increase.

Table 3: SAT and UNSAT problem counts for different datasets over numbers of variables.

Number of Variables	**hard3sat**		**uniform**		**symbolic**	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
3	5	5	5	5	5	5
4	4	5	5	5	5	5
5	5	5	5	5	5	5
6	3	5	5	5	5	5
7	3	5	5	5	5	5
8	4	5	5	5	5	5
9	3	5	5	4	5	5
10	0	5	5	5	5	5
11	2	5	4	5	5	5
12	1	5	5	5	5	5
13	2	5	3	4	5	5
14	2	5	4	5	5	5
15	3	5	5	5	5	5
16	3	5	4	5	5	5
17	1	5	4	5	5	5
18	2	5	3	4	5	5
19	0	5	4	5	5	5
Total	43	85	76	82	85	85

Table 4: SAT and UNSAT problem counts for different datasets over various clause intervals.

Number of Clauses	**hard3sat**		**uniform**		**symbolic**	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
1-10	13	0	63	5	13	0
11-20	14	10	13	10	25	10
21-30	7	15	0	14	22	15
31-40	8	10	0	12	17	10
41-50	1	10	0	7	7	10
51-60	0	15	0	20	1	15
61-70	0	10	0	7	0	10
71-92	0	15	0	7	0	15
Total	43	85	76	82	85	85

Table 5: Mean (std dev) of grounded text lengths (in characters) and number of sentences for `hard3sat` and `uniform` datasets over different numbers of variables.

Number of Variables	**hard3sat**		**uniform**	
	#Characters	#Sentences	#Characters	#Sentences
3	1304.5 (466.5)	9.1 (3.11)	268.2 (35.7)	3.0 (0.0)
4	1834.3 (641.2)	12.89 (4.91)	1488.9 (967.3)	9.9 (6.64)
5	2294.7 (805.4)	15.5 (5.87)	1801.3 (1356.7)	11.0 (8.07)
6	2865.8 (1002.7)	19.88 (7.12)	2342.7 (1724.9)	13.6 (9.67)
7	3409.6 (1214.1)	23.62 (7.46)	2810.3 (2184.3)	16.0 (12.1)
8	3843.1 (1336.7)	25.89 (9.83)	2912.4 (2193.0)	17.0 (12.54)
9	4314.1 (1137.4)	31.12 (9.49)	3456.1 (2904.4)	18.67 (14.87)
10	6198.0 (306.9)	42.0 (0.0)	5462.1 (3882.0)	23.8 (16.68)
11	5959.4 (1559.8)	40.14 (10.01)	5986.4 (4676.2)	26.78 (20.1)
12	6661.3 (1473.9)	47.0 (9.8)	7020.9 (5572.6)	29.1 (23.07)
13	6802.6 (1913.8)	46.71 (14.2)	7050.4 (4745.3)	31.14 (21.75)
14	7399.1 (1900.8)	51.57 (12.7)	8676.7 (6560.6)	36.56 (27.51)
15	7630.4 (2397.2)	51.5 (15.91)	7867.7 (5724.2)	32.3 (23.44)
16	8281.9 (2652.6)	55.25 (18.09)	10058.0 (7357.4)	40.33 (28.27)
17	9896.7 (2319.7)	66.17 (14.29)	9184.4 (6702.8)	38.11 (26.97)
18	9885.4 (2886.4)	65.43 (18.14)	11502.4 (8580.7)	45.71 (34.67)
19	11750.2 (831.2)	80.0 (0.0)	12344.6 (9099.6)	51.11 (36.87)

Table 6: Mean (std dev) of grounded text lengths (in characters) and number of sentences for the `hard3sat` and `uniform` datasets over various clause intervals.

Number of Clauses	**hard3sat**		**uniform**	
	#Characters	#Sentences	#Characters	#Sentences
1-10	1141.0 (237.9)	7.62 (1.5)	1203.0 (729.2)	5.96 (2.36)
11-20	2179.0 (512.0)	14.92 (3.06)	2678.9 (472.8)	14.04 (2.87)
21-30	3721.4 (496.8)	25.45 (3.05)	4383.9 (668.5)	25.21 (2.81)
31-40	5113.5 (373.9)	35.72 (2.24)	7474.1 (1360.7)	36.5 (3.15)
41-50	6479.6 (465.5)	43.82 (2.09)	10764.4 (758.7)	44.0 (2.38)
51-60	7878.7 (650.3)	55.0 (3.38)	13627.3 (1420.0)	56.3 (3.11)
61-70	9738.5 (587.5)	65.5 (2.64)	16016.9 (1938.3)	65.71 (3.25)
71-92	11375.7 (729.1)	76.0 (3.38)	20353.4 (1778.9)	83.71 (6.85)

B.4.1 Themes Used for Text Grounding

We did the text grounding of the `hard3sat` dataset using the broad topic, “banking". We did the text grounding of the `uniform` dataset using themes (LLM generated) related to banking, such as Banking Technology, Customer Experience in Banking, Customer Services in Banking, Customer Support in Banking, Digital and Mobile Banking, Financial Literacy, Financial Planning, Financial Planning and Advisory, Financial Services, Investment Banking and Services, Investment Strategies, Loan and Credit, Mobile Banking, Online Banking, Personal Finance, Savings and Investments, Savings and Loans, etc.

B.5 Correlation of accuracy with consistency

Table 7 reports the Pearson correlation coefficient (r) and p-value between average LLM accuracy and the percentage of `consistent` ground truth samples for each experiment and LLM, across input groups defined by the number of clauses. n indicates how many such input groups were included in each calculation, when grouped by the number of clauses.

Table 7: Correlation between LLM accuracy and the percentage of `consistent` inputs, grouped by the number of clauses. n indicate the total number of input groups formed based on clause count. r represents Pearson correlation, and p-value denotes statistical significance.

experiment	hard3sat	hard3sat	symbolic	symbolic	uniform	uniform
llm	GPT-4o	o3-mini	GPT-4o	o3-mini	GPT-4o	o3-mini
r	0.76	0.88	0.93	0.89	-0.50	0.35
p-value	<0.001	<0.001	<0.001	<0.001	<0.001	0.0086
n	41	41	46	46	56	56

For the `symbolic` and `hard3sat` datasets, both models exhibit strong positive correlations ($r > 0.75$), indicating that their accuracy increases when the ground truth contains more logically consistent problems. This suggests a systematic bias: LLMs are more likely to correctly classify queries when the majority class is `consistent`. On the `uniform` dataset, `GPT-4o` shows a moderate negative correlation, while `o3-mini` shows a weak positive correlation, implying that the relationship between accuracy and ground truth consistency is less straightforward for this dataset. The statistical significance (p-values) confirms that these correlations are strong. Overall, these findings suggest that LLMs may be less accurate when faced with `inconsistent` inputs. The plots in Section 4.3 further confirm this bias, demonstrating the systematic tendency of LLMs to predict `consistent` across the `symbolic` and `hard3sat` datasets. Given that `symbolic` is not textually grounded unlike `hard3sat`, this bias is present irrespective of the input being textual or symbolic.