Strategies for Improving NL-to-FOL Translation with LLMs: Data Generation, Incremental Fine-Tuning, and Verification

Anonymous ACL submission

Abstract

Logical reasoning is a fundamental task in natural language processing that presents significant challenges to Large Language Models (LLMs). While symbolic representations such as first-order logic (FOL) are well suited for logical reasoning, translating natural language (NL) into FOL often results in errors that are underexplored. We address this by categorizing the FOL translation errors in LLMs for deductive reasoning task and propose methods to improve translation quality, specifically for small (7B) language models. We introduce PROOFFOL, a high-quality FOL-annotated subset of ProofWriter dataset created using GPT-4o. The models fine-tuned on this silver standard data outperform large (70B) language models. Additionally, for better data utilisation in data-scarce settings, we present an incremental framework that combines data augmentation with a novel symbolic translation verification. Augmentation generates additional training data by splitting (premises, conclusion) pairs, which when used for fine-tuning results in improved performance over the model fine-tuned on the original data. Our investigation of the translation errors leads to generation of a perturbation dataset consisting of simulated NL-to-FOL translation errors and their corresponding corrections, which is used to train a verifier to identify and correct potential syntactic and semantic FOL translation errors. Our approach leverages limited human-annotated data, achieving state-of-the-art results on the ProofWriter and ProntoQA datasets. [1]

1 Introduction

Recent state-of-the-art methods for logical reasoning from natural language (NL) descriptions operate via translation (Pan et al., 2023; Ye et al., 2024; Olausson et al., 2023). In these methods, a large language model (LLM) is tasked to translate statements from NL to first-order logic (FOL), which is then sent for execution to external Satisfiability modulo theories (SMT) solvers such as Z3 (De Moura and Bjørner, 2008) and Prover9 (McCune, 2005). Recent work (Yang et al., 2024) has highlighted the systematic errors that even the most advanced LLMs (such as GPT-4) make during translation of a *single* NL statement into its corresponding FOL. Realistic logical reasoning scenarios are significantly more demanding, as they involves multiple premise statements followed by a conclusion to be verified. These scenarios require consistent NL-to-FOL translations (e.g., in predicate naming or the translation of logical operators) across multiple statements. However, very little is explored on the pattern of syntactic and semantic errors LLMs make in such translation scenarios.

Existing approaches to reducing NL-to-FOL translation errors have limited impact. They rely on the LLM's ability to understand and self-correct the translation inaccuracies based solely on the error message from the external SMT solver (Pan et al., 2023). However, the ability to comprehend such error messages is often restricted to larger-scale models (e.g., 175B), and is generally unavailable in smaller LLMs (e.g., 7B, 13B). Furthermore, while syntactic translation mistakes (e.g., *misplaced operator:* $P \wedge \rightarrow Q$, or *missing quantifiers:* $P(x) \rightarrow Q(x)$) result in runtime errors detected by the tools, many semantic errors (e.g., *use of incorrect quantifiers: All men are mortal.* $\exists x(Man(x) \rightarrow Mortal(x))$) bypass SMT tools without triggering any runtime error. This limitation hinders such approaches from correcting less trivial errors. A straightforward solution to reduce errors is to fine-tune smaller models on a large-scale NL-to-FOL translation data. However, existing datasets offer very little support, with FOLIO (Han et al., 2022) as the only FOL-specific human-annotated dataset containing around $1k$ ex-

[1] The code for fine-tuning, augmentation and verification, and PROOFFOL dataset are attached with the submission.

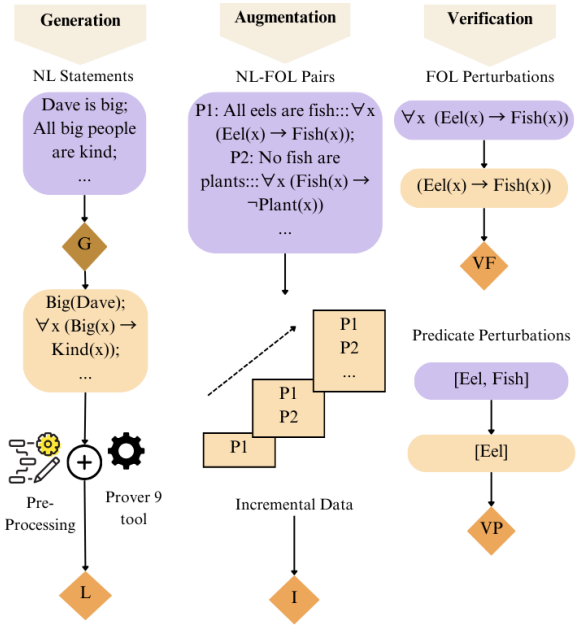


Figure 1: Training Pipelines. Generation: Natural Language (NL) statements are translated into First-Order Logic (FOL) statements using a generator (G). These FOL statements are filtered and pre-processed to extract a refined subset (ProofFOL), which is used to fine-tune a smaller language model (L). Augmentation: The NL-FOL pairs are iteratively expanded to generate a larger dataset. For each NL-FOL pair (P), data is incrementally split and expanded. This augmented dataset is utilized to train another smaller language model (I). Verification: Errors observed in the FOL statements and predicates are used to create a perturbation datasets. These datasets are leveraged to train two types of verifiers: an FOL Verifier (VF) for correcting FOL statement errors, and a Predicate Verifier (VP) for handling predicate-specific issues.

amples of NL (premises, conclusion) and their FOL translations. MALLS (Yang et al., 2024), another synthetic dataset, contains $28k$ pair of single statements and FOL translations. Even with access to larger-scale fine-tuning data, a correction mechanism for smaller models remains essential to catch both syntactic and semantic errors on-the-fly during inference.

In this paper, we propose methods to reduce NL-to-FOL translation errors. First, we address the lack of fine-tuning data by using GPT-4o to create PROOFFOL, a dataset of $10,424$ (premises, conclusion) pairs (extracted from ProofWriter (Tafjord et al., 2020)) and their FOL translations. These translations are validated using Prover9 to ensure correctness and passed through multiple formatting checks. Models fine-tuned on PROOFFOL, such as LLaMA-2 13B (Touvron et al., 2023) and Mistral 7B (Jiang et al., 2023), outperform larger baselines such as LLaMA-2 70B and Mixtral 8×7B in translation quality and logical reasoning tasks on ProofWriter and ProntoQA (Saparov and He, 2022).

Second, to effectively utilize the scarce but high-quality human-generated data (i.e., FOLIO), we propose a set of incremental techniques. Specifically, to increase the number of training instances, we split each record into multiple datapoints in an incremental fashion and then train the model to produce the predicates, and to generate the FOL for each premise and conclusion statement one-by-one. This approach improves predictive accuracy on FOLIO by 41%. Finally, to enable fine-grained correction of semantic and syntactic errors during inference, we train separate models (i.e., T5 (Raffel et al., 2020)) to verify predicates and FOLs on-the-fly, and apply necessary corrections when needed. Using simulated errors from perturbed FOLIO ground truths, these models either verify correctness or provide corrections, yielding an additional 17% improvement on FOLIO. This process is detailed in Figure 1.

Our findings highlight that data is crucial for surpassing the current performance limits of several LLMs, particularly when employing more accessible models for logical reasoning tasks. Our data generation pipeline allows us to create PROOFFOL, the largest FOL-annotated logical reasoning dataset to this date. Our incremental training presents an effective data augmentation method particularly useful for data scarce conditions, while the verifier mechanism and corresponding training protocol offers a promising pathway to verify correctness of symbolic forms generated by LLMs at inference time.

2 Related Work

LLM for symbolic translation The use of formal language translations by LLMs was initially attempted by Nye et al. (2021), with an intent to emphasize the importance of dual process theory for logical reasoning tasks. Following this, the process of reasoning was offloaded to theorem provers and LLMs served as systems to generate symbolic translations (Pan et al., 2023; Ye et al., 2024; Olausson et al., 2023). The available research in this method majorly differs in variation of formal language (used by different theorem provers) and verification process to handle translation errors. These

methods make use of expensive and ambiguous[2] GPT models, restricting the symbolic framework to non-critical domains. Corresponding to the work in formal logic, Yang et al. (2024) applied supervised finetuning to LLaMA model to improve the natural language to first-order logic translations at a sentence level. Our research shifts the focus to building a complete translation system that can handle multiple statements. In addition to this, we perform a systematic analysis of translation errors which enabled us to build a verification mechanism.

Symbolic Decoding with LLMs The choice of decoding strategies can improve text generation, specifically in LLMs where the output follows a structured format. In neuro-symbolic models, neuro-logic decoding (Lu et al., 2020, 2021) applies symbolic constraints. Interactive theorem provers were also used alongside LLMs to ensure a constrained generation of the reasoning path (Poesia et al., 2023). Other techniques like contrastive step-wise decoding helped with improving the probability of a correct reasoning path (Su et al., 2023). The success of these symbolic decoding strategies motivates our research to apply verification during the inference stage.

Deductive reasoning benchmarks Deductive logical reasoning requires logical derivation of conclusion using a set of premises. Benchmarks such as ProofWriter (Tafjord et al., 2020) and ProntoQA (Saparov and He, 2022) highlight multi-hop reasoning paths, but limit their use in tasks requiring formal rigor [3]. To address this, we build an FOL dataset to adapt ProofWriter for formal reasoning tasks, enhancing its ability to handle structured, interpretable logical inferences and enabling access to verification and theorem proving. The capability of this dataset when compared to existing benchmarks is detailed in Appendix B.

FOLIO (Han et al., 2022), with human-annotated FOL sentences, offers a semantically complex deductive task but is limited by its small size, limiting its use for developing or improving formal language translation models. We present data augmentation technique to overcome this issue. To the best of our knowledge, our work is the first at using the incremental setting (augmentation), and verification in the context of logical reasoning with NL.

[2]We refer to ambiguity in the data used to train the GPT model.

[3]Although ProofWriter dataset is generated using logic programs, the original logical forms of ProofWriter data and framework used for conversion to natural language are not released with their data and therefore not available.

3 NL-to-FOL Translation Errors

First-order logic (FOL) is a logical framework using variables, functions, and quantifiers, often applied in natural language reasoning. Large Language Models (LLMs) have demonstrated varying success in translating natural language into formal representations. Among these formalism, NL-to-FOL translation presents unique challenges involving syntactic and semantic interpretations. We attempt to categorize these syntactic and semantic errors as a foundation for our data perturbation protocol to train FOL verifiers (presented shortly).

Syntactic errors arise from deviations in grammatical rules during translation. For example, the rule *"every free variable assigned to a predicate must have a quantifier"* applied to the statement "Green people are blue" requires requires a quantifier '$\forall$' for the variable x. Missing quantifiers or operators can cause parsing errors, detected by tools like Prover9, which provides feedback on syntax issues. Common categories include:

- *Parsing errors:* Missing/invalid operators or parentheses.
- *Type errors:* Missing quantifiers or sentence-level discrepancies.
- *Token errors*: Use of invalid tokens (e.g., $).

Semantic errors are harder to detect as they conform to formal structure but misrepresent meaning. For example, the statement "All rabbits have fur" translated to the FOL $\exists x$(Rabbit(x) $\rightarrow$ Have(x, fur)) incorrectly quantifies the statement. Key categories include:

- *Sense errors:* General inaccuracies in NL-to-FOL mapping.
- *Arities errors:* Predicate mismatches in argument count.

These errors are further analyzed in detail (see Appendix H).

4 Incremental Fine-Tuning and Verification

4.1 Data Generation and Fine-tuning

The alignment of a language model to follow instructions for a specific task can be accomplished by fine-tuning on substantial data. The task of formal translations require first-order logic of their

natural language counterparts. Ideally, this task is at a passage level rather than sentence level, which makes it challenging for a language model to follow a required format. To overcome this, we need sufficient passage level translations, which are time-consuming to generate through human annotations. We introduce a streamlined process for generating this FOL data and ensuring correctness of the format, grammar, and order of the translations.

Data Generation For the data generation process, we pick ProofWriter which comes with large number of training records, each consisting of multipe premises and a conclusion, and variations in depth of reasoning. The format of the FOLs is set to be consistent with the "Prover9" theorem prover, which has a human understandable, formal language syntax.[4] The fixed format of demonstrations is adapted from Pan et al. (2023), where we generate predicates followed by first-order logic statements for each sentence. We standardize the formal language to Prover9 and apply it consistently across all datasets.

At a glance, the output from the GPT model has formatting issues, such as assigning numbers to each generation, explaining the task before generation, and solving for conclusion after producing translations. These issues are parsed using pattern matching to obtain the maximum number of translations. After the parsing stage, the syntax check is done by Prover9, where the tool can provide unique feedback for each form of syntax error. When analysing these errors, we observed grammar rules that can be fixed in the tool to include unicode decoding, allow unordered quantifiers and support negation of a full formula. This addition of grammar rules minimized the penalization for translations. The semantic errors were identified by comparing the ground truth label from ProofWriter with Prover9-generated output based on FOLs. This systematic pipeline enabled the retention of 70% of the silver-standard data. We name this dataset PROOFFOL, comprising $10,424$ NL-FOL passage-level pairs of NL-FOL translations.

Supervised Fine-tuning (SFT) Each input x to the SFT is a set of premise statements $P_x(=\{P_1, P_2, \ldots, P_n\})$, a conclusion C_x, and an instruction (I) represented as $[P_x, C_x, I]$. In order to avoid over-fitting the model to certain spurious patterns in GPT-4o translations, we include human-generated dataset (FOLIO) in the mix. We create a set of models built on the full dataset, providing a perspective for model behaviour with size of the dataset. Since this SFT model employs both existing and synthetic data, it imposes a dilemma of the effect of gold-standard data on the results when compared to the generated silver-standard. To validate the reliability of our generated data, we fine-tune the model on a subset of the dataset and examine how increasing the data size impacts logical reasoning tasks. For a given input x, the output of SFT models are predicates of x (denoted as $Pred_x$), and FOLs of its premises and conclusion, $[Pred_x, FOL_{P_x}, FOL_{C_x}]$.

4.2 Incremental Techniques

The SFT method described in the previous section uses the FOLIO dataset as just a small component of the overall approach. With high number of records associated with PROOFFOL, we can assume that the in-distribution dataset will show a major improvement when compared to FOLIO. To enhance the performance of FOLIO dataset (and in general any similar data-scarce scenario), we introduce *'Incremental Techniques'* for maximizing the use of limited data. These techniques encompass data augmentation, incremental fine-tuning and inference, and incremental verification of predicates and FOLs, creating a comprehensive setup for FOL generation. We further expand this method to a smaller subset of PROOFFOL to simulate data scarce environment.

4.2.1 Data Augmentation

Supervised Fine-tuning a decoder model is technically an unannotated form of training as the *supervised* part of the fine-tuning refers to the label that is passed with the input. During a vanilla SFT process, we pass the whole output along with the input, and the model performs inference as a text completion task. This motivates our data augmentation method, where the model examines smaller part of the output rather than training on the whole output at a time. The FOL translation is one such task where the output sequence is lengthy and the model can deviate from generating the correct syntax.

To initiate the data augmentation process, we split the output of the original record to repre-

[4] Given the requisite for diversity in syntax and semantic, we first chose a few combinations of demonstrations, and ran a small scale experiment through GPT-4o for each combination. We then selected the most optimal demonstrations with the least format and translation issues in the resulting generated FOL data. We report these final demonstrations in Appendix C.

sent incremental data, where the first output is $[Pred_x]$, the second output is $[Pred_x, FOL_{P_1}]$, and so on till we reach the full output $[Pred_x, FOL_{P_1}..., FOL_{P_n}, FOL_{C_x}]$. This splits a single record into $n + 2$ records, where n is the number of premises and '$+2$' is for predicate and conclusion generation. The input remains the same for all the records. This data augmentation increases the dataset size to about $7\times$ and $20\times$ for FOLIO and ProofWriter, respectively. With this enhanced data, we train a set of SFT models for FOL translation tasks.

4.2.2 Inference

The SFT for augmented data is performed using two instructions, indicating two types of tasks. The first instruction is *"Generate predicates for the given natural language sentences."* to generate the predicates, and the second is *"Given a premise and conclusion, generate the first order logic form of the premises and conclusion."* to generate the FOL statements. At inference, we provide the model with the instruction to generate the predicates. These predicates, along with the input, are then passed to the model to infer the FOL statements. We categorize FOL inference into two forms.

- **Vanilla Inference:** In vanilla inference, the model is provided with the natural language statements and is asked to generate the predicates and the FOL translations for the complete input.
- **Incremental Inference:** In incremental inference, we first generate the predicates, and then limit the generation to a single FOL translation at a time by setting a low `maximum_new_token` parameter. For every FOL generation, we pass the previously generated values as input. For example, if we are generating FOL_{P_3}, the input would be $[P_x, C_x, Pred_x, FOL_{P_1}, FOL_{P_2}]$.

4.2.3 Verification and Correction

Since the incremental inference allows individual predicate and FOL generation, we train two verifiers; predicate and FOL, to detect and correct the potential errors. The term 'verifier' refers to both the verification and correction processes.

- **Predicate Verifier** The Predicate verifier takes the $[P_x, C_x, Pred_x]$, as the input, and evaluates the predicted predicate $Pred_x$. If the verifier considers $Pred_x$ correct, the output is just 'correct', otherwise the verifier will generate the corrected predicates set. To achieve this outcome, the verifier is trained on the perturbed predicates of the training dataset used for SFT. The perturbations for predicates are created based on the semantic errors related to predicates. Few perturbations used are; omitting predicate, omitting variable, adding variable, and adding duplicate predicate.
- **FOL Verifier** The FOL verifier takes $[Pred_{\text{NL}}, FOL_{\text{NL}}, \text{NL}]$ as the input, where FOL_{NL} represents predicted FOL form of a single premise (P_i) *or* conclusion (C_x) statement (represented as NL). The verifier evaluates FOL_{NL} as 'correct' or will generate a corrected FOL. Similar to predicate verifier, we used the original SFT training set and applied perturbations to train this Verifier model. The perturbations used in this method are crafted based on the common syntax and semantic errors in translations (Section 3). Few perturbation used in this verifier are; changing quantifier position, omitting a quantifier, omitting parenthesis, tweaking negations, and replacing operators.

In addition to these perturbation instances, we incorporate real errors from the SFT training data. Specifically, we run the inference of the training data on the SFT model and collect the predicate and FOL, which do not match the ground truth, as errors. We add these errors to the fully-controlled created perturbations to form the 'incorrect verifier training instances'. Finally, in order to verify 'correct' predictions, we take the predicate and FOL values from the ground-truth and label them as 'correct'. We generate the perturbation dataset from a seed of $1k$ examples of FOLIO and similarly for ProofWriter. For detail statistics on the size of the resulting perturbation datasets, please see Appendix J.

Inspired by Han et al. (2023), after the creation of the training data for the two verifiers, we finetune a T5-Large (Raffel et al., 2020) model to work as the verifier. We train two separate models for the Predicate and FOL verification on each dataset. During incremental inference, the verifier is incorporated into the decoding phase, as shown in Fig 2.

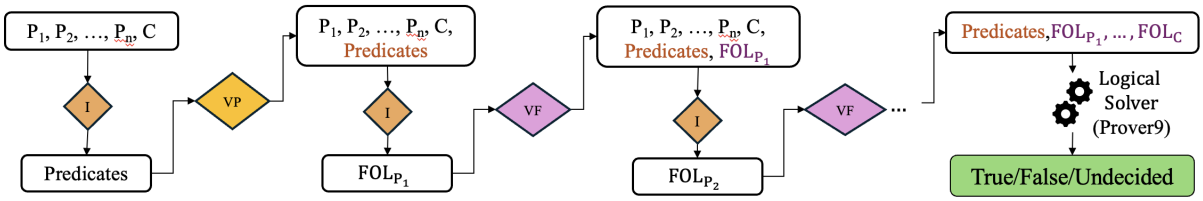


Figure 2: The overall flow of the incremental generation and verification at inference time. Here P_is and C denote Premises and Conclusion. For predicate and FOL generation, the output is run through the corresponding verifier.

5 Experimental Setup

PROOFFOL Data Generation. We used ProofWriter Open World Assumption (OWA) training data with depth 5 and generated 15,000 FOL records via `gpt4-o`. Regex parsing was applied to extract clean FOL statements, filtering incomplete or excessive text. The FOL statements are then passed to the Prover9 tool to perform deductive reasoning, where the tool returns 'True', 'False', 'Unknown', or 'None' for errors. Mismatches with ground truth were removed. See Appendix A for details of prompt templates and statistics.

Vanilla SFT. We fine-tuned LLaMA-2 (13B) and Mistral (7B) on ProofWriter, FOLIO, and ProntoQA using 3-epoch supervised fine-tuning (SFT) with LoRA (Hu et al., 2022) and 8-bit quantization for inference. Models were evaluated in two modes: NL-based (Standard) and FOL-based, leveraging Prover9. The initial baselines use in-context learning (ICL) for standard and FOL generation. The few-shot examples for standard generation are randomly sampled from the training data, ensuring a balanced output distribution. ProntoQA served as an out-of-distribution (OOD) benchmark with ProofWriter demonstrations during inference. Appendix C provides details on few-shot templates.

Incremental SFT. Incremental SFT is performed on LLaMA-2 13B model following the same training process as vanilla SFT, with adjustments for inference to reduce token generation time. Data augmentation increased data size of FOLIO (1,000 to $\sim$ 7000 records) and ProofWriter (1,000 to $\sim$ 20000 records). Fine-tuning was also performed on original datasets for baseline comparisons. Dataset statistics are in Appendix I.

Verifier Training. We trained four T5-large verifiers on perturbed datasets for FOLIO and ProofWriter. The perturbations were applied on their respective training data and vary with the complexity of the FOL statements. The T5 models were trained for 10 epochs using AdamW (Loshchilov and Hutter, 2019) and a learning rate of $5 \times e^{-5}$. Once trained, the verifiers could run in synchronous (online) or asynchronous (offline) mode. While the online mode corrects errors at each step of inference, before moving to next step (i.e., correction at time step t impacts step $t+1$ during generation), the offline mode applies corrections on the fully generated predicate and FOLs as a post-processing step (i.e., correction at time step t does not have any consequential effect on $t+1$). For time overhead of incremental decoding and verification, see Table 2.

6 Results and Discussion

6.1 Main Results

Table 1 presents results on logical reasoning dataset using in-context learning (ICL) and supervised fine-tuning (SFT), comparing performance across model sizes. LLaMA-2 70B and Mixtral 8×7B serve as baselines for LLaMA-2 13B and Mistral 7B, respectively.

Standard experiment reports free-form reasoning, where the LLM is given a question (premises and a conclusion) and is tasked to produce a direct response. Few-shot ICL results vary across datasets, with Mistral and Mixtral outperforming other models. LLaMA-2 13B fine-tuned on PROOFFOL achieves notable improvements for ProofWriter but shows limited gains for FOLIO and ProntoQA. Mistral models perform better after fine-tuning, particularly with smaller datasets.

FOL experiments show correlation of performance with model size, where, for few-shot ICL, LLaMA-2 70B and Mixtral perform consistently better than their smaller versions. ProofWriter achieves 86% accuracy with LLaMA-2 13B model, fine-tuned on PROOFFOL, which is a significant

	Type		Model	Standard	FOL
ProofWriter	ICL	n-shot	LLaMA-2 70B	41.83	78.33
			LLaMA-2 13B	44.16	24.66
			Mistral 7B	49.67	66.50
			Mixtral 8 × 7B	45.83	85.00
	SFT	5000	LLaMA-2 13B	64.16	53.50
			Mistral 7B	70.33	**98.17**
		10000	LLaMA-2 13B	**85.66**	86.33
			Mistral 7B	69.50	97.83
ProntoQA	ICL	n-shot	LLaMA 70B	50.60	38.80
			LLaMA-2 13B	47.19	11.4
			Mistral 7B	50.60	10.80
			Mixtral 8 × 7B	58.40	13.00
	SFT	5000	LLaMA-2 13B	55.40	53.20
			Mistral 7B	60.00	78.60
		10000	LLaMA-2 13B	53.20	47.80
			Mistral 7B	**70.40**	**85.40**
FOLIO	ICL	n-shot	LLaMA-2 70B	50.74	34.97
			LLaMA-2 13B	43.84	24.13
			Mistral 7B	51.23	35.96
			Mixtral 8 × 7B	57.14	**42.36**
	SFT	5000	LLaMA-2 13B	40.89	26.11
			Mistral 7B	**67.98**	26.11
		10000	LLaMA-2 13B	40.89	34.48
			Mistral 7B	66.01	27.59

Table 1: Comparison of models' deductive reasoning accuracy under Standard and FOL-based output prediction. Here ICL denotes "in-context learning" with n-shots (details of shots in appendix), and SFT denotes "supervised fine-tuning" (on $5k$ or $10k$ training data subset from PROOFFOL). Accuracy metrics in **bold** signify notably high performance within the same benchmark, while underline indicates best results under Standard and FOL for ICL and SFT.

gain in performance when compared to ICL. Mistral fine-tuned models are the state-of-the-art in FOL generation for ProofWriter datasets, where the model produces 0 syntax errors after fine-tuning. ProntoQA, an altered form of ProofWriter, shows similar trends in performance gain with our fine-tuned models. Mistral 7B fine-tuned on PROOFFOL data outperforms all the ProntoQA baselines. For ProntoQA, there is an observed negative effect of overfitting, when LLaMA-2 is trained on larger dataset (increasing training data from $5k$ to $10k$), but we don't notice this for Mistral. We speculate this might be reflective of difference in model size and how it impacts potential training memorization (i.e., overfitting) for larger models.

FOLIO, as expected, is a challenging dataset with complex language and structure. Mixtral model achieves the highest few-shot accuracy at 42%. Fine-tuned LLaMA-2 13b improves from 24% to 34%, and is on-par with a much larger

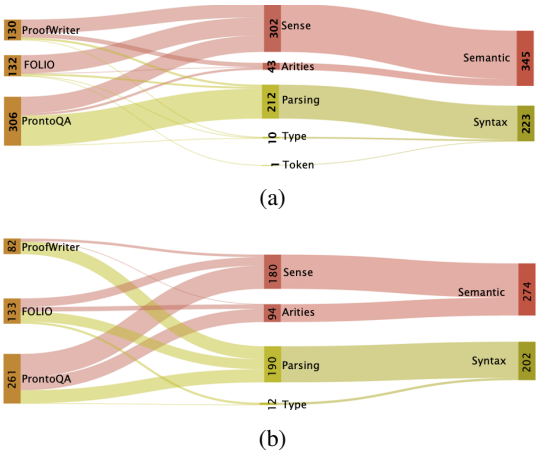


Figure 3: Error distribution of LLaMA-2 70B (top) and 13B (bottom) post fine-tuning on PROOFFOL.

LLaMA-2 70B. Syntax and semantic error metrics are detailed in Appendix G.

Error Distribution We provide an error distribution of models on NL-to-FOL translation errors. The results for LLaMA-2 70B with ICL and LLaMA-2 13B after fine-tuning is provided in Fig 3. ProofWriter and ProntoQA show decline in errors, whereas FOLIO stays equivalent to the LLaMA-2 70B ICL model. This plot indicates that a model significantly smaller in size can achieve better generation over a larger model when fine-tuned with relevant data.

6.2 Incremental Results

Incremental techniques are performed on ground truth FOLIO and a subset of PROOFFOL dataset. The intention behind the incremental technique is to show that the data augmentation method improves the translations over the original dataset. In Table 2, we focus on the shift in performance between LLaMA-2 13B model fine-tuned on the original 1000 records and augmented $1000 \times n$ records, where n represents the scale at which the data grows after augmentation. Both FOLIO and ProofWriter have low performance in FOL generation when trained with a small dataset, but FOLIO improves steadily with incremental techniques. This pushes us to use a verifier to further the performance. With the predicates corrected during generation and FOLs corrected after inference, this verifier based incremental setting achieves 37% using Offline verifier, which outperforms LLaMA-2 70B ICL accuracy 34.97% (Table 1). ProofWriter, when trained incrementally, provides varied results with different inferences. The verifier inference

Models	Inference	FOLIO	ProofWriter
ICL Baseline	**Vanilla**	24.13	24.66
SFT Vanilla ($1k$)	**Vanilla**	22.66 (01:55)	24.50 (03:36)
SFT Incremental ($1k^*$)	**Incremental**	32.02 (02:58)	27.16 (02:42)
SFT Incremental ($1k^*$)	**+Verifier (On-Off)**	**37.44** (03:05)	29.05 (03:10)
SFT Incremental ($1k^*$)	**+Verifier (On-On)**	29.56 (03:27)	**32.50** (03:31)

Table 2: Comparison of LLaMA-2 13B models on FOLIO and ProofWriter with various training and inference protocols. $1k^*$ is the incremental SFT data that expands to $7k$ (FOLIO) and $20k$ (ProofWriter) data points for training. Inference time for SFT models for each record is reported as average (MM:SS). **Vanilla**: Predicates and FOLs are generated in one pass. **Incremental**: Predicates are generated in one pass and then fed into the next step along with premises and conclusions to generate FOLs one by one. **+Verifier (On-Off)**: Predicate verifiers are called during inference (Online) while FOL verifiers are called once all FOLs are generated (Offline). **+Verifier (On-On)**: Both verifiers are used online.

model achieves 29% an 32% accuracy when compared to the SFT model on original dataset, 24%. It can be noted that the performance of the FOLIO model is lower when the FOL verifier is online, since any error from the verifier is passed on to the next generation and can cause a domino effect. The change in inference times between incremental and verifier settings is minimal. [5]

6.3 Ablation Studies

The incremental methods follows certain rules of finetuning and generation. Our model uses two instructions for predicate and FOL generation. We performed an ablation study on using different variation of incremental training and inference for FOLIO dataset. The type of ablations and their performances are given in Table 3. These were performed before training the verifier module. In place of the verifier, we initially used Prover9 to check for syntax and any invalid generation followed a sampling by the LLM, and the first error free FOL was selected. This method proved ineffective as the sampling method was time-consuming. The results in the table are after the tool verification, except for the *Check* model, where we use LLM as a verifier.

Mixed is the current method of incremental training and inference, where we use different instructions for predicate and FOL generation. *Single* method uses only one instruction;'Complete the generation'. This performs equivalent to the *Mixed*, but results in additional syntax errors, presumably

[5]For ProofWriter, adaptive token sizing reduces inference time by adjusting token size based on sentence length.

Instruction	Accuracy	Syntax errors
Mixed	35.46	64
Single	35.47	70
Ordered	32.02	63
Unique	21.18	80
Check	27.09	108

Table 3: Comparison of Incremental Methods with Instruction Fine-tuning. Each method represents the type of instruction used for fine-tuning and inference. The syntax errors are out of 203 test records in FOLIO.

because of the vague instruction. *Ordered* uses different instructions for each FOL generation. The instructions carry information about the sentence that is required to be translated. This method helped keep the syntax errors low, but lowered the overall performance. Instead of passing the previously generated FOL to the next generation, we applied *Unique*, where the statements are split and passed one at a time with the predicate values. This performs poorly and does not include passage level translations. In *Check*, we use the perturbation dataset with specific instructions along with the training dataset for fine-tuning. We instruct the model to identify and correct the errors at inference. This method relies on the language model to perform a new task with limited data and proves to be ineffective.

7 Conclusion

Formal language translation systems for logical reasoning task worked effectively in the era of LLMs, with persisting translation errors. In this paper, we provided an understanding of general translation errors by LLMs, when used as NL-to-FOL translation systems. We highlighted the importance of first-order logic (FOL) ground truth data and present a pipeline to generate high quality FOLs for ProofWriter dataset, introducing an FOL-annotated data called PROOFFOL. Using PROOFFOL, we fine-tuned a set of smaller language models and showed an increase in performance over larger LMs. Additionally, the issue of data scarcity is addressed via proposing incremental techniques, which cover data augmentation, inference verification, and correction. Our experiments on 3 benchmarks highlight the potential of our proposed framework.

8 Limitation

The incremental setup proposed in our method depends on the quality of the data. While FOLIO is highly effective, it may occasionally introduce some noise. The silver data generated using GPT-4, though processed with care, may contain minor inconsistencies, as detailed in Appendix D. Additionally, using models like GPT-4 can present challenges in reproducing the exact dataset for future reference. To address this, we have provided all necessary prompts to facilitate consistent generation of similar datasets.

The baselines for the provided datasets are run similarly to the existing work, though we acknowledge that exact comparisons are not possible due to the use of different models and some challenges with reproducibility. As a result, we consider outputs from larger models as the baseline, following a similar methodology. We also present results from larger proprietary models in Appendix F; however, due to limited information on their training process, we treat these comparisons as supplementary rather than definitive.

We would also like to acknowledge that incremental methods may result in longer inference times for the Llama model; however, from our previous experiments on vanilla SFT, we noticed that inference times vary depending on the models used.

References

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

Jiuzhou Han, Nigel Collier, Wray Buntine, and Ehsan Shareghi. 2023. Pive: Prompting with iterative verification improving graph-based generative capability of llms. *arXiv preprint arXiv:2305.12392*.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, et al. 2022. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, et al. 2021. Neurologic a* esque decoding: Constrained text generation with lookahead heuristics. *arXiv preprint arXiv:2112.08726*.

Ximing Lu, Peter West, Rowan Zellers, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Neurologic decoding:(un) supervised neural text generation with predicate logic constraints. *arXiv preprint arXiv:2010.12884*.

Man Luo, Shrinidhi Kumbhar, Mihir Parmar, Neeraj Varshney, Pratyay Banerjee, Somak Aditya, Chitta Baral, et al. 2023. Towards logiglue: A brief survey and a benchmark for analyzing logical reasoning capabilities of language models. *arXiv preprint arXiv:2310.00836*.

William McCune. 2005. Release of prover9. In *Mile high conference on quasigroups, loops and nonassociative systems, Denver, Colorado*.

Maxwell Nye, Michael Tessler, Josh Tenenbaum, and Brenden M Lake. 2021. Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning. *Advances in Neural Information Processing Systems*, 34:25192–25204.

Theo X Olausson, Alex Gu, Benjamin Lipkin, Cedegao E Zhang, Armando Solar-Lezama, Joshua B Tenenbaum, and Roger Levy. 2023. Linc: A neurosymbolic approach for logical reasoning by combining language models with first-order logic provers. *arXiv preprint arXiv:2310.15164*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore. Association for Computational Linguistics.

Gabriel Poesia, Kanishk Gandhi, Eric Zelikman, and Noah D Goodman. 2023. Certified deductive reasoning with language models. *arXiv preprint arXiv:2306.04031*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Abulhair Saparov and He He. 2022. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*.

Ying Su, Xiaojin Fu, Mingwen Liu, and Zhijiang Guo. 2023. Are llms rigorous logical reasoner? empowering natural language proof generation with contrastive stepwise decoding. *arXiv preprint arXiv:2311.06736*.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. 2020. Proofwriter: Generating implications, proofs, and abductive statements over natural language. *arXiv preprint arXiv:2012.13048*.

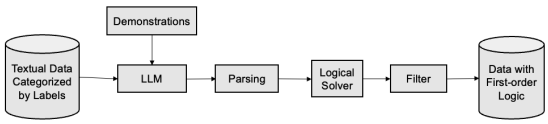
Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. 2024. Harnessing the power of large language models for natural language to first-order logic translation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6942–6959, Bangkok, Thailand. Association for Computational Linguistics.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2024. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36.

A Data Generation

The data pipeline applied for generating ProofFOL is detailed in Fig 4. The train data is 15000 data points with depth-5 from ProofWriter dataset. The 15k records are sampled randomly ensuring a fair distribution of the labels; True, False, and Uncertain. The LLM here is GPT4o with a output token length of 1000 for each generation. We use $url = /v1/chat/completions$ format for batch generations of GPT4o. The logical solver is Prover 9, a theorem prover suitable to run in python environment with nltk library, that uses CNF conversions, quantifier operations, and skolemization to transform the clauses into a tree format. Parsing errors by Prover 9 occur when the FOL formula cannot be converted to a tree structure because it does not follow specific grammar rules. After filter and parsing stages, we get 10424 records with FOL statements. Syntax errors are the errors thrown by the tool and semantic errors are measured by comparing the ground truth label with the solver output.



Data	Total Train	Syntax Errors	Output Mismatch	Final Train
ProofWriter	15,000	3881	695	10,424

Figure 4: Overview of the Data Generation Pipeline and Key Statistics

B Dataset Comparison

The *ProofFOL* dataset consists of around $10,000$ NL-FOL translation data-points, consisting of premises and conclusions. This makes it the largest existing FOL dataset designed for logical reasoning tasks. Specifically, in the context of deductive reasoning tasks (Luo et al., 2023), current datasets often present natural language statements with either a conclusion that must be logically deduced from the premise or multiple-choice options evaluating the logical relevance of a reading comprehension passage (Liu et al., 2020). We focus on single-answer questions that enable rule-based deduction. Such rule-based deduction facilitates application in formal settings.

ProofFOL distinguishes itself from other datasets in terms of data size, the availability of a comprehensive training set, formal translations,

	FOLIO	LogiQA	ProntoQA	ProofWriter	ProofFOL
Large Dataset	×	✓	✓	✓	✓
Training Set	✓	✓	×	✓	✓
FOL translations	✓	×	×	✓	✓
Rule-Based Generation	✓	~	✓	✓	✓

Table 4: Comparison of Deductive Reasoning Datasets with Our Dataset, **ProofFOL**. A large dataset is defined as having over 5,000 records. The training set indicates the availability of data for fine-tuning. Ground truth FOL translations refer to provided First-Order Logic expressions. Rule-based generation denotes the ability to derive solutions systematically using predefined rules without external intervention.

and its ability to support rule-based derivations. Figure 4 highlights these variations across the primary logical deductive reasoning datasets.

C Few Shot Examples

In this section, the few-shot format used for all the in-context learning tasks are presented.

Few-shot for Data-augmentation The Fig 5 shows the format of the few-shot example for data generation. We tried this with 2 examples for 50 unique training data points, but it did not generate better translation results. Instead, we ensured that all operators are covered in this example, specifically negation. We first generate predicates associated with the sentence. This is followed by FOL generation. Each FOL generation comes with the natural language statement and this format is adapted from (Pan et al. 2023). We use the same format of generation for all of our experiments with the exception of Predicates description, as this part is harder to verify. The description is removed from ProofFOL.

- Standard generation: For standard generation, we randomly sample examples from the training set. We use 5-shot for FOLIO and 4-shot for ProofWriter and ProntoQA as shown in Fig 6 and Fig 7
- FOL generation: For FOL generation, we partially use examples from (Pan et al. 2023) and sample the rest from the training set. The FOL syntax is fixed to represent Prover9 format. We use 3-shot for FOLIO and 2-shot for ProofWriter and ProntoQA as shown in Fig 8 and Fig 9

D Human evaluation of ProofFOL

The ProofFOL dataset is a silver-standard dataset. To ensure the accuracy of the data, we conducted

You are an expert who works with theorem provers. Given some context and a question, generate the predicates and the first-order logic formula for contexts and question. Here is an example.

Context:
Anne is quiet. Erin is furry. Erin is green. Fiona is furry. Fiona is quiet. Fiona is red. Fiona is rough. Fiona is white. Harry is furry. Harry is quiet. Harry is white. Young people are furry. If Anne is quiet then Anne is red. Young, green people are rough. If someone is green then they are white. If someone is furry and quiet then they are white. If someone is young and white then they are rough. All red people are young.
Question:
Anne is white.
###
Predicates:
Quiet(x) ::: x is quiet
Furry(x) ::: x is furry
Green(x) ::: x is green
Red(x) ::: x is red
Rough(x) ::: x is rough
White(x) ::: x is white
Young(x) ::: x is young
Premises:
Quite(Anne) ::: Anne is quiet.
Furry(Erin) ::: Erin is furry.
Green(Erin) ::: Erin is green.
Furry(Fiona) ::: Fiona is furry.
Quite(Fiona) ::: Fiona is quiet.
Red(Fiona) ::: Fiona is red.
Rough(Fiona) ::: Fiona is rough.
White(Fiona) ::: Fiona is white.
Furry(Harry) ::: Harry is furry.
Quite(Harry) ::: Harry is quiet.
White(Harry) ::: Harry is white.
∀x (Young(x) → Furry(x)) ::: Young people are furry.
Quite(Anne) → ¬Red(Anne) ::: If Anne is quiet then Anne is not red.
∀x (Young(x) → Rough(x)) ::: Young, green people are rough.
∀x (Green(x) → Rough(x)) ::: Young, green people are rough.
∀x (Green(x) → White(x)) ::: If someone is green then they are white.
∀x (Furry(x) ∧ Quite(x) → ¬White(x)) ::: If someone is furry and quiet then they are not white.
∀x (Young(x) ∧ White(x) → Rough(x)) ::: If someone is young and white then they are rough.
∀x (Red(x) → Young(x)) ::: All red people are young.
Conclusion:
White(Anne) ::: Anne is white.

<INPUT>
###

Figure 5: Few shot example with instruction for Data Generation process

a human evaluation on a random sample of 50 records from our dataset (containing a total of $1,018$ statements) where the final outputs were correct. Out of these $1,018$ translations, we identified only 22 errors (i.e., 2%), half of which were minor and did not alter the meaning of the FOL statements. Another extrinsic indication of data quality is the improvement reported in Table 2 for SFT on 5k and 10k records. Additionally, we verified other semantic aspects, including predicate values and variable completions, finding zero errors in predicate values and only one error in variable completions. Further details on this can be found in the Table 6.

Given a premise, you have to reason whether the conclusion is True, False or Uncertain.
Context: The Metropolitan Museum of Art is a museum in NYC...
Conclusion: Mary lives in Manhattan.
Answer: Uncertain

Context: All artificial satellites are important scientific achievements...
Conclusion: All important scientific achievements are U.S. inventions.
Answer: False
FS3
FS4
FS5
<INPUT>
Answer:

Figure 6: Standard Generation Few-shot examples for FOLIO: For brevity, we highlighted the format and the instructions and minimized the content in the few-shot examples.

Given a premise, you have to reason whether the conclusion is True, False or Uncertain.
Context: Charlie is cold. ...
Question: Harry is not furry.
Answer: Unknown

Context: Anne is green. ...
Question: Bob is not young.
Answer: False
FS3
FS4
<INPUT>
Answer:

Figure 7: Standard Generation Few-shot examples for ProofWriter and ProntoQA

E Datasets

We utilize three datasets in our experiments.

- ProofWriter: The training set for ProofWriter is sampled from the depth-5 records. For test-set, we use the one provided in (Pan et al. 2023)
- ProntoQA: ProntoQA is a test dataset. We use ProofWriter examples as training set and test-set sample from (Pan et al. 2023)
- FOLIO: FOLIO has 1001 training set records and 203 dev set records. We use the original training set for SFT models and dev for evaluating the model.

Dataset	Standard	FOL
ProofWriter	4	2
FOLIO	5	3
ProntoQA	4	2

Table 5: Number of Few-shot examples used for creating baselines in Table 1

Given some context and a question, generate the predicates and the first-order logic formula for contexts and question. Here are the examples.

Context:
All people who regularly drink coffee are dependent on caffeine. ...
Question:
Rina is either a person who jokes about being addicted to caffeine or is unaware that caffeine is a drug.
###
Predicates:
Dependent(x) ::: x is a person dependent on caffeine. ...
Premises:
∀x (Drinks(x) → Dependent(x)) ::: All people who regularly drink coffee
Conclusion:
Jokes(rina) ⊕ Unaware(rina) ::: Rina is either a person who jokes about being addicted to caffeine or is unaware that caffeine is a drug.

FS2
FS3

<INPUT>
###

Figure 8: FOL Generation Few-shot examples for FOLIO

Error Types	Description	Number of Errors
FOL Match	Validates the correctness of a single FOL statement in relation to its corresponding NL text.	22
Non-subject Predicates	Ensures that predicates are meaningful and do not include any shortcuts.	0
Non-subject Constants	Flags instances where constants take shortcuts, similar to predicate validation.	1

Table 6: Error types and descriptions for human evaluation of a sample of ProofFOL dataset, specifically 1018 translations.

Dataset	Original	Augmented
ProofWriter	1000	7288
FOLIO	998	20145

Table 7: Size of training datasets before and after data augmentation

F Additional Baselines

We conducted experiments on additional proprietary models, and the results are presented in Table 8. The results demonstrate that our model, trained on **ProofFOL**, outperformed these closed source models in ProofWriter, highlighting the significance of our dataset.

G SFT Models

SFT with PROOFFOL is performed on two models; LLaMA-2 13B and Mistral 7B. These models are selected based on their low computational cost and model parameters. The models are fine-tuned for 3-

Given some context and a question, generate the predicates and the first-order logic formula for contexts and question. Here is an example.

Context:

Anne is quiet. Erin is furry. ...

Question:

Anne is white.

###

Predicates:

Quiet(x) ::: x is quiet...

Premises:

Quite(Anne) ::: Anne is quiet. ...

Conclusion:

White(Anne) ::: Anne is white.

FS2

<INPUT>

###

Figure 9: FOL Generation Few-shot examples for ProofWriter and ProntoQA

Dataset	Model	Standard	FOL
ProofWriter	Llama2 70B	41.83	78.33
	Mixtral 8X7B	45.83	85.00
	GPT-4o mini	36.33	74.17
	Gpt-3.5-turbo-instruct	52.83	**90.17**
	Gemini Flash	**54.33**	89.00
ProntoQA	Llama2 70B	50.74	34.97
	Mixtral 8X7B	57.14	42.36
	GPT-4o mini	57.40	53.80
	Gpt-3.5-turbo-instruct	49.40	24.80
	Gemini Flash	**72.20**	**93.60**
FOLIO	Llama2 70B	50.74	34.97
	Mixtral 8X7B	57.14	42.36
	GPT-4o mini	40.39	45.81
	Gpt-3.5-turbo-instruct	50.25	47.29
	Gemini Flash	**64.04**	**55.66**

Table 8: Performance comparison across other proprietary baselines.

epochs with a fixed batch size and LoRA (Hu et al., 2022). The inference is done using an 8-bit quantization with the trained LoRA adaptor. Temperature and `top_p` are fixed at 0 and 1 respectively, with a variation of `maximum_new_token` value based on the dataset. ProofWriter requires larger number of tokens at inference when compared to FOLIO because of the size of input. Additionally, as NL-based baseline, separate versions of models are also fine-tuned on textual data without symbolic translations (the corresponding results are reported under Standard). The test data used in our experiments are taken from Pan et al. (2023) for ProofWriter and ProntoQA. FOLIO comes with a pre-defined dev dataset, that is used for evaluations.

Table 9 is an extension of Table 1 from the main paper. It shows the variations in the syntax and semantic errors with ICl and fine-tuning. There is a constant trend of lower syntax errors with fine-tuning the models with relevant data for both Llama and Mistral models.

	Type		Model	Syntax	Semantic
ProofWriter(600)	ICL	n-shot	LLaMA-2 70B	44	86
			LLaMA-2 13B	314	138
			Mistral 7B	130	71
			Mixtral 8 × 7B	44	46
	SFT	5000	LLaMA-2 13B	71	15
			Mistral 7B	0	11
		10000	LLaMA-2 13B	69	13
			Mistral 7B	0	13
ProntoQA(500)	ICL	n-shot	LLaMA 70B	203	103
			LLaMA-2 13B	266	177
			Mistral 7B	420	26
			Mixtral 8 × 7B	413	22
	SFT	5000	LLaMA-2 13B	108	126
			Mistral 7B	26	81
		10000	LLaMA-2 13B	139	122
			Mistral 7B	13	60
FOLIO(203)	ICL	n-shot	LLaMA-2 70B	19	113
			LLaMA-2 13B	95	59
			Mistral 7B	95	35
			Mixtral 8 × 7B	79	38
	SFT	5000	LLaMA-2 13B	103	47
			Mistral 7B	64	86
		10000	LLaMA-2 13B	88	45
			Mistral 7B	48	99

Table 9: Syntax and Semantic error count for FOL generation

H Error Analysis

The NL-FOL translation errors can be identified using the tool feedback or a mismatch in tool output with the ground truth. We identify these issues and categorize them into syntax and semantic errors. Fig 10 shows examples of each type of error. These are identified by manually analysing the FOL statements in cases where there was no feedback from the tool. The details of each error are provided here.

- Missing quantifier: When a predicate includes a variable, it must have either a Universal Quantifier '∀' or an Existential Quantifier '∃'. This error occurs if either quantifier is missing.
- Parenthesis error: The formula becomes invalid if there is an extra or a missing parenthesis.
- Completion error: The FOL is either incomplete or contains additional text that disrupts its logical flow.

- Quantifier location: Quantifiers that are either repetitive or incorrectly positioned result in grammatical inaccuracies in the expression.
- Missing variable: When multiple quantifiers are present, the tool fails to parse predicates that lack free variables.
- Special token: The tool does not handle special characters in the input.
- Unknown operator: The tool does not support parsing mathematical equations.
- Predicate error: These errors arise when predicates are reused with different subjects, omitted entirely, or conjoined inappropriately, leading to erroneous interpretations of the logical constructs in FOL statements. Such misinterpretations can affect the accuracy and reliability of responses generated by LLMs.
- Incorrect quantifier: This occurs when an existential quantifier is incorrectly chosen in situations that require a universal quantifier, leading to a logical contradiction between the terms. This mismatch between the chosen quantifier and the necessary logical condition can result in flawed reasoning and inconsistencies in logical analysis.
- Predicate mismatch: It occurs when LLMs are tasked with generating predicates based on text passages and fail to recognize synonymous terms as equivalent. This results in the tool counting synonyms as distinct tokens, leading to discrepancies in predicate generation.
- Arities error: It occurs when predicates are inconsistently applied with a varying number of constants across different statements. Such discrepancies can introduce ambiguity in logical inferences. This is a semantic error that is captured by the tool.
- Subject predicate: The logic is flawed when a subject is used both as a predicate and a constant in the expression.

I Data Augmentation

Data augmentation for FOL generation uses incremental data assignment, where the output is divided into multiple tasks, as shown in Fig 10. This method is applied on two datasets; ProofWriter and FOLIO. A subset of the ProofWriter dataset is extracted from the ProofFOL data and passed for data augmentation, increasing the size to 20X the original. For FOLIO, we take the full dataset and perform augmentation, making it 7X larger. Given the style of incremental data, we remove few records from FOLIO dataset which do not follow the one-to-one mapping of text and FOL. The size stats are detailed in Table 7.

Vanilla		
X	Input	Output
X^i	Generate FOL [$\mathbf{P_1}$, $\mathbf{P_2}$, …, $\mathbf{P_n}$, **C**]	[Pred, P_1^{FOL}, P_2^{FOL}, P_3^{FOL}, C^{FOL}]

Incremental		
X_0^i	Generate predicate [$\mathbf{P_1}$, $\mathbf{P_2}$, …, $\mathbf{P_n}$, **C**]	Pred
X_1^i	Generate Premise FOL [$\mathbf{P_1}$, $\mathbf{P_2}$, …, $\mathbf{P_n}$, **C, Pred**]	P_1^{FOL}
⋮		
X_n^i	Generate Conclusion FOL [$\mathbf{P_1}$, $\mathbf{P_2}$, …, $\mathbf{P_n}$, **C, Pred**, $\mathbf{P_1}^{FOL}$, $\mathbf{P_2}^{FOL}$, …, $\mathbf{P_n}^{FOL}$,]	C^{FOL}

Figure 10: Data augmentation: the vanilla represents original data point. Incremental is the data that is present after augmentation.

J Verification Perturbations

The perturbations used for training T5 verifier models are designed from the errors in the previously discussed error analysis. The perturbations are different for FOLIO and ProofWriter dataset, as FOLIO is a complex dataset with additional operators when compared to ProofWriter. To handle the complexity of FOLIO dataset, we pass the training set data to the SFT model and match the translations with ground truth. Any mismatch is treated as a perturbation. Other perturbations are based manually included. This dataset has a portion of correct values, making it a verification and correction system. The count of these datasets is detailed in Table 11.

FOLIO Perturbations The predicate perturbations are designed based on the commonly observed errors in the predicates. These are not complex errors, but usually a missing predicate or variable. Based on this, we use three types of perturbations.

- Omit One Predicate: We randomly omit a predicate from all predicates.
- Omit One Variable: We choose a predicate with the maximum number of variables and omit the last one in the chosen predicate.
- Omit Both Variable and Predicate: We choose a predicate with the maximum number of variables, omit the last argument in the chosen predicate, and also randomly omit a predicate from the rest of the predicates.

The FOL perturbations tackle the syntax errors that are common while generating FOL for FOLIO test data.

- Change Quantifier Position: We move the quantifier position to different positions in a FOL statement.
- Omit One Quantifier: We randomly omit one quantifier from a FOL statement.
- Omit Last Bracket: We omit the last bracket in a FOL statement.

ProofWriter Perturbations The ProofWriter dataset has simpler, but larger number of predicates. based on this, we design 4 types of perturbations.

- Omit one predicate: We randomly omit a predicate from all predicates.
- Omit or add one variable: We choose a random predicate, and if the predicate consists of multiple variables, we omit one, and if it consists of only one variable, we add one.
- Add plural predicates: To handle the synonymous predicate issue, we randomly select a predicate, create a plural form, and add it to the set of predicates.
- Duplicate predicate: We select a random predicate and add a variable to it. This is added back to the set of predicates.

In addition to the quantifier, FOL generations in ProofWriter consists of three major operators; and, imply, and negation. We include variations in this for our perturbation data. ProofWriter can be divided into simple FOL statements (facts) and complex ones (rules). The simple statements usually do not contain any operators or quantifiers. Based on this, we design 5 types of perturbations.

- Add or omit negations: We randomly select facts and either add a negation or remove an existing one.
- Omit arguments from facts: We randomly select predicates from facts and omit the variables if the predicate has more than one variable.
- Add or omit quantifier: We randomly select rules and either add a quantifier or remove an existing one.
- Swap operators: We replace 'and' with 'imply' operator or vice-versa if the operator exists in the FOL statements.
- Add plural predicates: To handle the synonymous predicate issue, we randomly select a predicate, create a plural form, and add it to the set of predicates.

Incremental and Verification Ablation We present additional ablation done on the incremental methods and analyse their results.

- Incremental Ablation: The incremental finetuning uses two instruction; generate predicates and generate FOL (brief). This allows us to perform a full scale inference rather than an incremental one, by increasing the output token size. We first let the model predict the predicates and the input along with the predicates is passed to the LLM for full FOL generation. Since the data in the augmented set consist of such a case (last value in augmentation), the model is able to generate the FOL in a flow. We performed inference in this manner and the results are in Table 12, where vanilla* represents the full FOL inference. The results in ProofWriter are higher than the incremental results, but on closer observation, we determine that the model hallucinates few cases in the vanilla* and helps with better performance. The incremental method allows a strict format of generation and forces the model to generate the FOL for one statement at a time, without adding additional values. Additionally, because of this mismatch in number of statements, we cannot use a FOL verifier on vanilla*.

K System Requirements for Experimentation

The Llama 2 model weights are downloaded from the official Llama website https://llama.meta.com/llama-downloads/. Mistral-7B (https://huggingface.co/mistralai/Mistral-7B-v0.3) model and Mixtral-8x7B(https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1) are accessed via the Hugging Face interface. All the models are gated and require access, which is typically a safety measure and can be easily granted. The transformer version used in our experiments is $v4.40.0$ and should ideally be above $v4.28.0$ or the latest version to run Llama and Mistral models without errors. All other library requirements will be specified in the code.

	Type	Example
Parsing	Missing Quantifier	BerkeleyCollege($\underline{x}$) ∧ResidentialCollegeAt($\underline{x}$, yaleUniversity)
	Parenthesis error	BeneficialTo(cherry, people) $\oplus$On(cherry, warningList)$\underline{)}$ → ¬RedFruit(cherry)
	Completion error	$\forall x$(Athlete(x) →¬NeverExercises(x)) Never: does not exist a time
Type	Quantifier location	$\underline{\exists y}$(Own(emily, y) ∧Roommate(y)) → $\underline{\exists y}$(Own(emily, y) ∧LiveIn(emily, apartment))
	Missing variable	$\forall x \underline{\exists y}$(In(indonesia) ∧ Prosecutor(x) ∧ SpecialCrime(y) → InvestigatePersonally(x, y))
Token	Special token	Endowment(yale, 42.3 billion)
	Unknown Operator	$\forall x$(Rating(x, y) ∧ $y \underline{\geq} 4 \rightarrow$ Listed(x))
Sense	Predicate errors	Error: ¬Solid2Pointers(jack) ∧ Successful3Pointers(jack) True: ¬GoodAt(jack, twos) ∧ GoodAt(jack, threes)
	Incorrect Quantifier	Error: $\underline{\exists x}$ (FleaBeetle(x) → ¬InFamily(x, chrysomelidae)) True: $\forall x$ (FleaBeetle(x) → ¬In(x, chrysomelidaeFamily))
	Predicate Mismatch	Error: ¬High(NewHaven); Low(towerA) True: ¬High(NewHaven); ¬High(towerA)
Arities	Arities errors	Error: Sees(Tiger, Mouse); $\forall x$(((Visits(x, Rabbit)) ∧ (Sees(Mouse))) → (Visits(x, Tiger))) True: Sees(Tiger, Mouse); $\forall x$ (((Visits(x, Rabbit)) ∧ (Sees(x, Mouse))) → (Visits(x, Tiger)))
	Subject Predicate	Platypus(platypus) ∧¬Teeth(platypus) ∧Mammal(platypus)

Table 10: Common Errors in First-Order Logic: The first block of errors are syntactic and the second are semantic errors. This table categorizes errors by their cause, with the *underlined* text highlighting the specific cause or location of each error. For semantic errors, there are 'True' values to make sense of the error.

Dataset	Verifier	Correct	Training	Manual
ProofWriter	Predicate	1978	326	2991
ProofWriter	FOL	2375	1595	2358
FOLIO	Predicate	1724	-	3448
FOLIO	FOL	2000	-	3241

Table 11: Amount of data for each type of perturbations

Inference	FOLIO	ProofWriter
vanilla*	31.52	<u>64.00</u>
Incremental	32.02	27.16
+Verifier	<u>37.44</u>	29.05

Table 12: Comparing two forms of inference for incremental models. This inference ablation is done on FOLIO and ProofWriter datasets.