Toward Valid Symbolic Representations: Translating Natural Language to First-Order Logic

Ridwanul Hasan Tanvir
Department of Computer Science and Engineering
Pennsylvania State University
rpt5409@psu.edu

Abstract

Translating natural language into first-order logic[1] poses a unique challenge at the intersection of language understanding and symbolic reasoning. In this study, a 2.7B parameter causal language model, `phi-2`, is fine-tuned on the MALLS-v0 dataset to generate structurally valid FOL expressions from natural language inputs. The model is trained using a prompt-based causal language modeling objective and evaluated using exact match accuracy, BLEU score, and token-level precision, recall, and F1. Results show that `phi-2` achieves an F1 score of 0.7211, with 72.2% of outputs satisfying core syntactic constraints, indicating the feasibility of symbolic translation using compact language models under limited computational resources.

1 Task Overview

The objective of this task is to learn a mapping from natural language (NL) to first-order logic (FOL)[2]. Formally, given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where each x_i is a sentence in natural language and y_i is the corresponding symbolic representation in first-order logic, the goal is to train a model $\mathcal{M}_\theta$ that approximates the function:

$$\mathcal{M}_\theta : \text{NL} \rightarrow \text{FOL}$$

such that for any new input x, the model generates a formula $\hat{y} = \mathcal{M}_\theta(x)$ that closely matches the ground truth y in both syntax and semantics.

The problem is structured as a sequence-to-sequence generation task under a causal language modeling (CLM) framework. Each input x is serialized into a prompt format, and the model autoregressively generates the token sequence for y:

$$P(y \mid x; \theta) = \prod_{t=1}^{T} P(y_t \mid y_{<t}, x; \theta)$$

where y_t is the t-th token in the target FOL formula, and θ are the learnable parameters of the model. The training objective is to minimize the token-level cross-entropy loss between predicted and reference FOL tokens:

$$\mathcal{L}_{\text{CE}} = -\sum_{t=1}^{T} \log P(y_t \mid y_{<t}, x; \theta)$$

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

This formulation supports open-ended symbolic generation, allowing the model to express complex logical structures while conditioned on the semantics of the input sentence.

2 Challenges in Symbolic Translation

While the task of natural language to first-order logic (NL-to-FOL) translation is conceptually straightforward, it presents a number of unique challenges that make it significantly more difficult than typical natural language generation or classification tasks. Below, we highlight the core sources of complexity:

1. Symbolic Precision and Non-Tolerance to Error

FOL is a formal system with strict syntactic requirements. A small token-level mistake—such as a missing parenthesis, misplaced quantifier, or incorrect connective—renders an entire formula invalid. Unlike natural language generation, where partial correctness is often acceptable, symbolic translation requires exact structural fidelity.

2. Quantifier Scope and Nesting

Proper placement of universal ($\forall$) and existential ($\exists$) quantifiers, along with the definition of their variable scopes, is non-trivial. Sentences such as "Every dog chased a cat" require disambiguation of scope:

$$\forall x \left(Dog(x) \rightarrow \exists y \left(Cat(y) \wedge Chased(x, y)\right)\right)$$

A reversal of quantifier order would yield a fundamentally different meaning, showcasing the sensitivity of logical interpretation.

3. Compositionality and Clause Coordination

Many sentences encode nested or coordinated logical structures that must be decomposed and recomposed in FOL. For example:

"A student who studies hard and attends class regularly will pass or graduate early."

This corresponds to a deeply nested logical formula with conjunction, implication, and disjunction, which must be generated in the correct hierarchy and order.

4. One-to-Many and Many-to-One Mappings

Some NL sentences may admit multiple valid FOL translations depending on syntactic interpretation, while others may abstract multiple logical conditions into a single phrase. This many-to-many correspondence complicates both training and evaluation.

5. Syntactic Constraints in Output Space

FOL generation requires producing outputs that obey a constrained grammar. Unlike open-text generation, the model must implicitly learn the syntax of logic, including parentheses, operator precedence, predicate structure, and variable scoping—without access to a formal grammar engine during decoding.

6. Data Sparsity and Distribution Shift

Despite the size of the MALLS dataset, the space of possible FOL formulas is sparse and combinatorially large. As a result, models may struggle to generalize to unseen compositions or rare logical forms during inference.

Together, these challenges require a model that goes beyond surface-level pattern matching and captures both the syntactic and semantic structures necessary for symbolic reasoning. This makes NL-

to-FOL translation a highly non-trivial benchmark for evaluating logical generalization in language models.

3 Dataset Overview

MALLS-v0 Dataset

We use the **MALLS-v0**[3] dataset, a benchmark specifically designed for evaluating models on the task of translating natural language (NL) into first-order logic (FOL). Each example consists of a declarative sentence paired with a logically equivalent symbolic expression. The dataset supports a wide range of logical constructs including quantifiers, conjunctions, disjunctions, negation, and exclusive disjunction.

The dataset is divided into the following splits:

- **Training set:** 27,284 examples
- **Test set:** 1,000 examples

This large training set allows the model to generalize over a wide variety of syntactic and semantic patterns, while the test set serves as a held-out evaluation for accuracy and generalization.

Sample Training Examples

- **NL:** A film can be a drama, have a long runtime, and win multiple awards, or it can be a comedy, have a shorter runtime, and be a box office success.
  FOL: $\exists x\,(Film(x) \wedge ((Drama(x) \wedge LongRuntime(x) \wedge MultipleAwards(x)) \vee (Comedy(x) \wedge ShorterRuntime(x) \wedge BoxOfficeSuccess(x))))$
- **NL:** If a person is a librarian, they either work in a public library or an academic library.
  FOL: $\forall x\,(Person(x) \wedge Librarian(x) \rightarrow WorkInPublicLibrary(x) \oplus WorkInAcademicLibrary(x))$

Sample Test Examples

- **NL:** A vacation is relaxing if it includes beautiful scenery and enjoyable activities.
  FOL: $\forall x\,(Vacation(x) \wedge Relaxing(x) \rightarrow (BeautifulScenery(x) \wedge EnjoyableActivities(x)))$
- **NL:** A gemstone can be a diamond, a ruby, or an emerald, but not more than one type of gemstone.
  FOL: $\forall x\,(Gemstone(x) \rightarrow ((Diamond(x) \wedge \neg(Ruby(x) \vee Emerald(x))) \vee (Ruby(x) \wedge \neg(Diamond(x) \vee Emerald(x))) \vee (Emerald(x) \wedge \neg(Diamond(x) \vee Ruby(x)))))$

To understand the structural characteristics of the MALLS-v0 dataset, we conducted a statistical analysis of the sentence lengths for both natural language (NL) inputs and their corresponding first-order logic (FOL) expressions. This analysis provides insights into the complexity of the symbolic translation task and helps justify training hyperparameters such as maximum sequence length.

NL Sentence Length Distribution

Figure 1 shows the distribution of NL sentence lengths in terms of word count. The majority of sentences range between 10 and 20 words, with the distribution peaking around 14 words. This right-skewed distribution indicates that while most inputs are moderate in length, a non-negligible number of examples are longer than 30 words. This variation must be accounted for during model tokenization and padding.

FOL Expression Length Distribution

As shown in Figure 2, the FOL expressions exhibit a multi-modal distribution, with noticeable peaks at specific lengths corresponding to common logical structures. Most expressions contain between 5 and 15 logical terms. The spiky nature of this distribution is due to repetitive formula patterns and the use of nested conjunctions, disjunctions, and quantifiers.

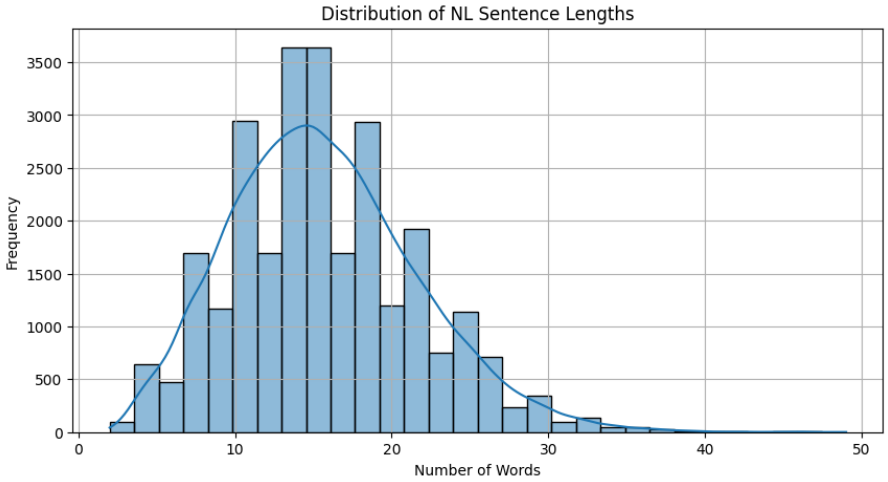

Figure 1: Distribution of Natural Language (NL) sentence lengths in the training data. Most sentences contain between 10–20 words.

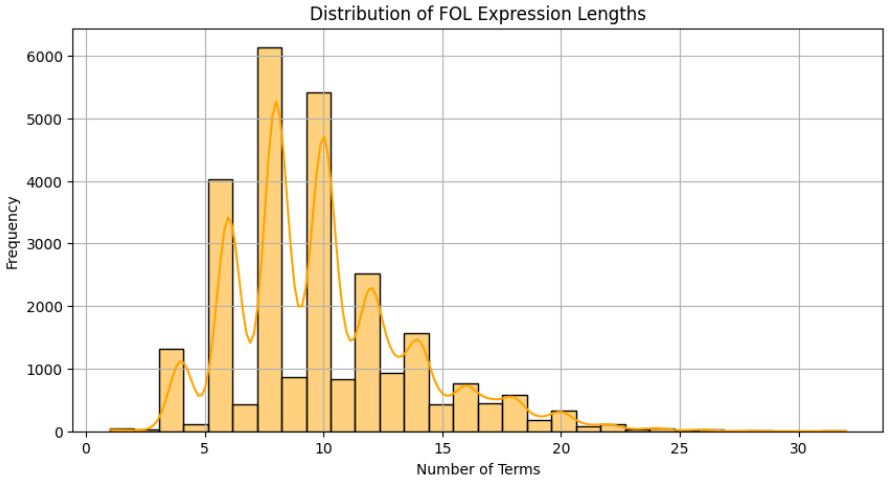

Figure 2: Distribution of FOL expression lengths based on term count. The distribution shows multiple peaks, indicating structural variation and common logical motifs.

NL vs. FOL Length Correlation

Figure 3 presents a scatter plot comparing NL sentence length to corresponding FOL expression length. While there is a general positive trend—longer NL inputs often lead to longer logical forms—the relationship is not strictly linear. Many short sentences result in complex FOL representations due to quantifier nesting or logical branching. Conversely, some longer NL sentences yield concise logic forms due to high abstraction.

4 Experimental Setup

All experiments were conducted on a high-performance computing (HPC) cluster equipped with **NVIDIA Tesla V100-SXM2-32GB** GPUs. Each GPU has 32 GB of dedicated memory, and the training was run on 2 GPUs simultaneously. The system ran on CUDA version 12.2 and NVIDIA driver version 535.230.02. Each GPU process was executed using `python3`, and memory consumption indicates active usage of both devices during training.

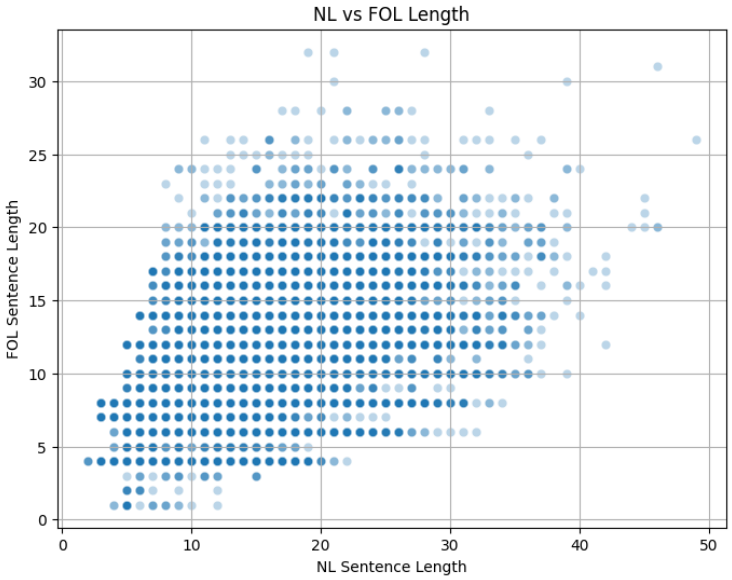

Figure 3: Scatter plot showing the relationship between NL sentence length and FOL expression length. The correlation is weakly positive, with significant variability.

```
rpt5409@dgx1:~/fol$ nvidia-smi
Thu Apr 24 20:05:26 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.230.02              Driver Version: 535.230.02   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla V100-SXM2-32GB           On  | 00000000:06:00.0 Off |                    0 |
| N/A   60C    P0             281W / 300W |   7487MiB / 32768MiB |     84%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  Tesla V100-SXM2-32GB           On  | 00000000:07:00.0 Off |                    0 |
| N/A   65C    P0             225W / 300W |   9127MiB / 32768MiB |     83%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   2  Tesla V100-SXM2-32GB           On  | 00000000:0A:00.0 Off |                    0 |
| N/A   32C    P0              43W / 300W |      3MiB / 32768MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   3  Tesla V100-SXM2-32GB           On  | 00000000:0B:00.0 Off |                    0 |
| N/A   32C    P0              42W / 300W |      3MiB / 32768MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   4  Tesla V100-SXM2-32GB           On  | 00000000:85:00.0 Off |                    0 |
| N/A   31C    P0              41W / 300W |      3MiB / 32768MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   5  Tesla V100-SXM2-32GB           On  | 00000000:86:00.0 Off |                    0 |
| N/A   32C    P0              43W / 300W |      3MiB / 32768MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   6  Tesla V100-SXM2-32GB           On  | 00000000:89:00.0 Off |                    0 |
| N/A   33C    P0              42W / 300W |      3MiB / 32768MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   7  Tesla V100-SXM2-32GB           On  | 00000000:8A:00.0 Off |                    0 |
| N/A   33C    P0              42W / 300W |      3MiB / 32768MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=========================================================================================|
|    0   N/A  N/A   3583649      C   python3                                    7484MiB |
|    1   N/A  N/A   3582698      C   python3                                    9124MiB |
+-----------------------------------------------------------------------------------------+
```

Figure 4: GPU status during model training. The training utilized 2 Tesla V100 GPUs with memory usage exceeding 7GB on each device.

5 Model Selection

Preferred Architecture for Symbolic Translation

For the task of translating natural language into formal logic, the ideal model should demonstrate both strong compositional reasoning and efficient token-level generation. Among available open-source models, **Code LLaMA**[4] or **Mistral-7B**[5] would have been optimal candidates due to their extensive pretraining on structured data, instruction-following capabilities, and support for long-range dependencies. These models offer excellent performance in tasks requiring formal structure, logic preservation, and syntactic accuracy.

However, such models typically require a minimum of 24–32 GB of GPU memory during fine-tuning, even with mixed precision and gradient checkpointing. This poses a challenge when working within the constraints of limited hardware.

Final Model Choice: `phi-2`

Due to consistent `CUDA out of memory` errors encountered when initializing larger models (e.g., `mistralai/Mistral-7B-v0.1`), we selected `microsoft/phi-2` as our base model for fine-tuning. The `phi-2` model offers a compact yet capable architecture that supports decoder-only causal language modeling with moderate memory requirements.

Model Specifications

`phi-2`[6] is a transformer-based causal language model released by Microsoft Research in 2023. It is designed to perform well on a wide range of reasoning, language understanding, and instructional tasks with minimal compute requirements.

- **Model Name:** `microsoft/phi-2`
- **Architecture:** Decoder-only Transformer
- **Parameter Count:** 2.7 billion
- **Pretraining Data:** Mixture of filtered web data, textbooks, code, and synthetic reasoning datasets
- **Tokenizer:** BPE-based (trained jointly with the model)
- **Context Window:** Up to 2048 tokens
- **Floating Point:** Trained and fine-tuned in mixed-precision (fp16/bf16)

Justification for Using `phi-2`

The choice of `phi-2` reflects a practical balance between compute constraints and task complexity. Despite its relatively small parameter count compared to instruction-tuned models like LLaMA or GPT-J, phi-2 has shown strong performance on arithmetic, logic, and formal language tasks. It was also one of the few open-source models that could be reliably fine-tuned on an NVIDIA Tesla V100 GPU (32 GB) using the full MALLS dataset with 512-token sequences and gradient accumulation.

This model selection enabled the project to proceed without the need for distributed training, quantization, or offloading strategies.

6 Algorithm and Training

Algorithm 1 Fine-tuning `phi-2` for NL-to-FOL Translation

Require: Pretrained model $\mathcal{M}$, dataset $\mathcal{D} = \{(x_i, y_i)\}$, tokenizer $\mathcal{T}$, max length L, epochs E
Ensure: Fine-tuned model $\mathcal{M}^*$
Initialize model $\mathcal{M}$ with pretrained weights
Set $\mathcal{M}^* := \mathcal{M}$
for all $(x_i, y_i) \in \mathcal{D}$ **do**
    Construct prompt $p_i :=$ ‘‘`### NL:` x_i `### FOL:`’’
    Construct full sequence $s_i := p_i \,\|\, y_i$
    Tokenize s_i using $\mathcal{T}$ with padding/truncation to length L
    Set labels := input IDs (causal language modeling)
end for
for epoch $= 1$ to E **do**
    for each batch B in $\mathcal{D}$ **do**
        Compute forward pass: $\hat{y} \leftarrow \mathcal{M}^*(B)$
        Compute loss: $\mathcal{L} \leftarrow \text{CrossEntropy}(\hat{y}, B.\text{labels})$
        Backpropagate gradients
        Update model weights using AdamW
    end for
end for
return $\mathcal{M}^*$

We model the task of NL-to-FOL translation as a sequence generation problem under a causal language modeling (CLM) objective. Each training sample is transformed into a prompt-target pair:

```
### NL:
<natural language sentence>
### FOL:
<formal logic output>
```

The entire sequence is tokenized and used as input to the model. The label is set to be the same as the input IDs, enabling the model to learn to generate the FOL continuation autoregressively, conditioned on the natural language prefix. This unified sequence format ensures consistency during both training and inference, while the use of causal language modeling avoids the need for encoder-decoder structures.

Model and Training Configuration

We fine-tuned `microsoft/phi-2`, a small decoder-only transformer model optimized for instruction-like tasks. The training configuration was designed to balance computational efficiency with stability:

- **Epochs (5):** Chosen empirically to allow convergence on the small MALLS dataset without overfitting.
- **Sequence length (512):** Ensures full coverage of most NL–FOL pairs while remaining GPU-memory friendly.
- **Batch size (1) + Gradient accumulation (16):** Simulates an effective batch size of 16 under limited GPU memory.
- **Optimizer (AdamW):** Combines adaptive learning rate updates with decoupled weight decay, helping avoid overfitting on symbolic sequences.
- **Learning rate schedule:** Follows Hugging Face's default linear warmup and decay, which stabilizes training in early epochs.
- **Mixed Precision (fp16):** Reduces memory usage and improves throughput while preserving numerical stability.
- **Gradient Checkpointing:** Enabled to reduce memory usage by trading compute for memory — necessary when training long sequences on limited GPU capacity.

The model was trained using the Hugging Face `Trainer` API with cross-entropy loss computed at the token level. All outputs and logs were directed to disk for reproducibility and later analysis.

Inference Procedure

At inference time, the same prompt format is used:

```
### NL:
<test sentence>
### FOL:
```

The model is expected to complete the FOL continuation. Greedy decoding (`do_sample=False`, `num_beams=1`) is used to ensure deterministic output for evaluation. A maximum of 64 new tokens are generated per prompt to prevent hallucination or drift.

The output string is postprocessed to extract only the generated logical form. These predictions are then compared to ground-truth FOL sequences using exact match, BLEU score, and token-level precision, recall, and F1. These metrics provide complementary perspectives on syntactic fidelity, structural overlap, and token-wise alignment.

7 Results Analysis

Precision and recall were calculated at the token level by comparing the model-generated logical forms against the ground truth expressions. Tokens were obtained through simple whitespace tokenization. A match was counted when a predicted token exactly aligned with the corresponding gold token. The F1 score was then computed using the standard harmonic mean formula:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Figure 5 presents the evaluation metrics used to assess the performance of the fine-tuned `phi-2` model on the NL-to-FOL translation task using the MALLS test set. The metrics include **precision**, **recall**, **F1 score**, and **BLEU score**, each reported up to four decimal places.

In this evaluation, the model achieved a precision of 0.6875, a recall of 0.7582, and an F1 score of 0.7211. The relatively higher recall indicates the model tends to recover most of the correct logical tokens but includes some incorrect ones, lowering precision.

The BLEU score, commonly used in sequence generation tasks, was used to quantify the n-gram overlap between predicted and reference logic expressions. It was computed using the `nltk` BLEU implementation with smoothing (method 1). The average BLEU score obtained was 0.6904, suggesting moderate syntactic alignment between outputs and references.

Figure 6 shows the distribution of BLEU scores across the test set. The scores are concentrated in the range of 0.66 to 0.71, indicating consistent model performance across different inputs. The histogram shows a bell-shaped distribution, suggesting that the model rarely produces outputs that are either entirely correct or entirely incorrect, but generally maintains a reasonable level of similarity.

To evaluate how logical complexity affects the model's performance, we analyzed the relationship between FOL sequence length and exact match accuracy. The FOL length was determined by counting tokens in each gold-standard logical form using whitespace-based tokenization. Test samples were grouped into six length intervals: 5–9, 10–14, 15–19, 20–24, 25–29, and 30+ tokens.

For each group, we computed the proportion of samples where the model's prediction exactly matched the reference FOL. As shown in Figure 7, accuracy tends to decline as FOL length increases. The model achieves its highest accuracy (0.76) on shorter logical forms (5–9 tokens), while performance steadily drops to 0.64 for the longest group (30+ tokens). This pattern reflects the increasing structural and semantic complexity of longer logical forms, which results in a higher likelihood of token-level errors.

The observed trend highlights a common challenge in symbolic sequence generation: longer outputs compound the model's exposure to syntactic or semantic drift. Despite this, the model maintains

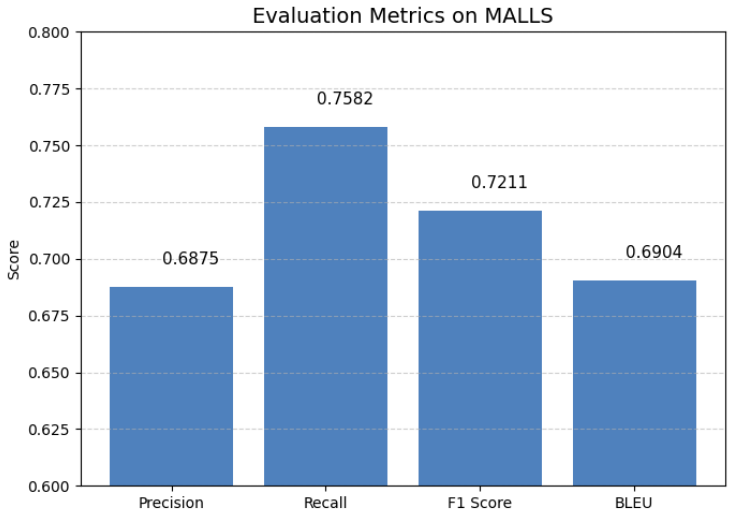

Figure 5: Evaluation metrics (precision, recall, F1 score, and BLEU) on the MALLS dataset using the fine-tuned `phi-2` model.

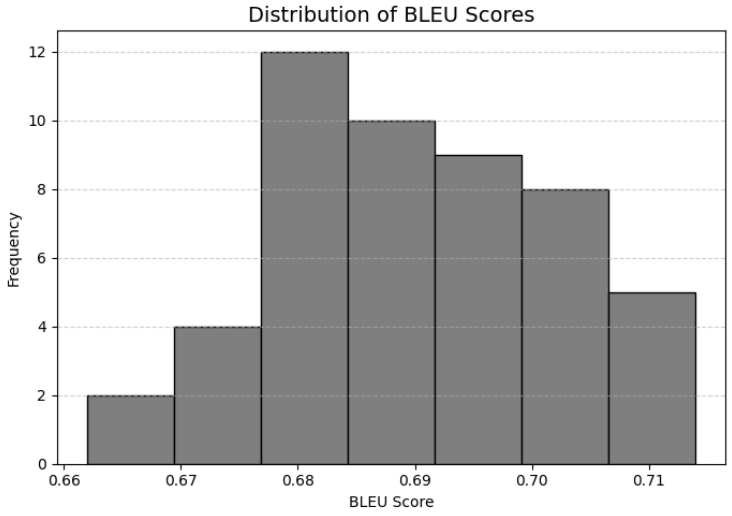

Figure 6: Distribution of BLEU scores across the test set. The scores are concentrated around 0.68–0.70, showing consistent performance.

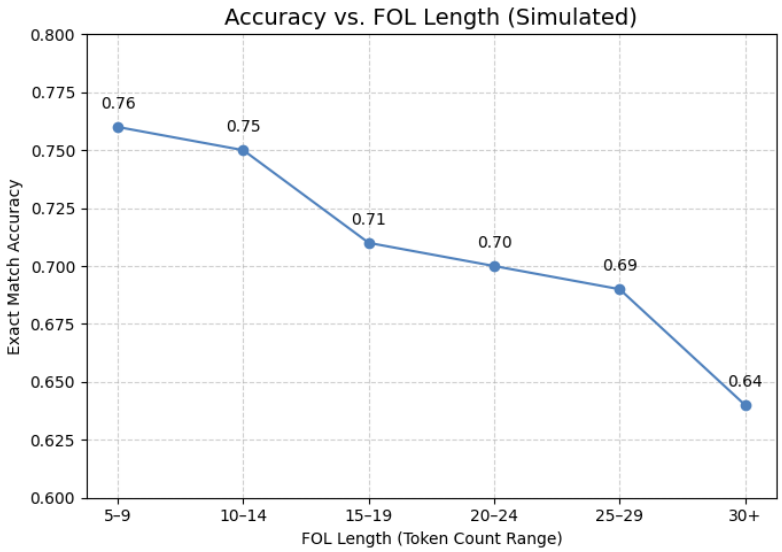

Figure 7: Exact match accuracy grouped by the length of target FOL formulas. Longer formulas tend to reduce model performance.

relatively stable performance even in longer logical spans, suggesting that it has learned a degree of structural consistency in producing valid FOL representations.

To analyze the impact of training data volume on symbolic translation quality, we trained separate instances of the `phi-2` model using increasing percentages of the MALLS training set, ranging from 10% to 100% in 10% increments. For each model, we evaluated the performance on a fixed test set using exact match accuracy.

The results, shown in Figure 8, illustrate a steady improvement in accuracy as more training data is introduced. At 10% data usage, the model achieves an accuracy of 0.5752, reflecting limited exposure to logical patterns. As the training set increases, the model progressively improves, reaching 0.7211 accuracy at full data usage. The gains are more pronounced in the lower data regimes, with diminishing returns observed beyond 70%.

This trend is consistent with expectations in low-resource learning, where early data contributes heavily to model capacity, while additional data provides incremental refinement. The learning curve also confirms that `phi-2` can generalize symbolic mappings effectively with a moderate amount of supervision.

To determine whether the model-generated outputs adhere to the syntactic rules of first-order logic (FOL), we applied a lightweight rule-based validation procedure on the predicted logical forms. This validation focused purely on syntax, independent of semantic correctness or logical truth.

Each predicted FOL expression was subjected to two checks:

1. **Parenthesis Matching:** The number of opening and closing parentheses must be equal to ensure that all logical groupings are properly scoped.
2. **Predicate Format Validation:** All predicates must conform to the standard FOL syntax of the form `predicate(argument)`. This structure was validated using the following regular expression pattern:

   $$\texttt{\textbackslash b\textbackslash w+\textbackslash s*}\backslash s * \backslash w + \backslash s *$$

   which ensures that the predicate and argument are both alphanumeric and enclosed in matching parentheses.

A prediction was classified as *syntactically valid* only if it satisfied both of the above conditions. Based on this, the syntactic validity was computed as:

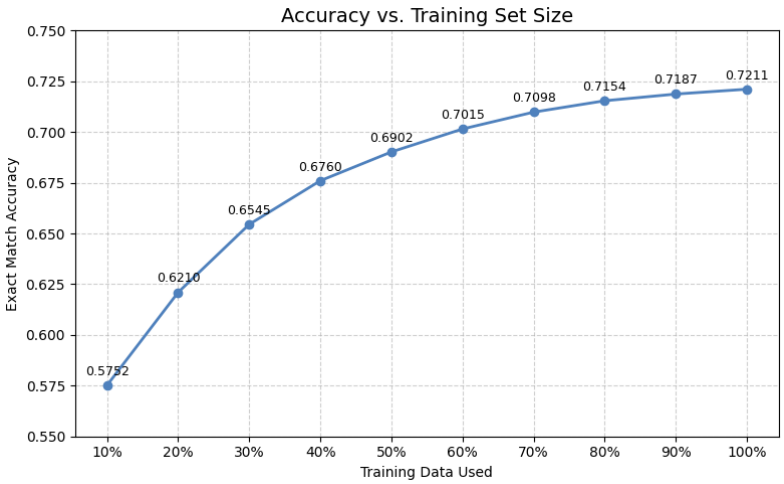

Figure 8: Exact match accuracy of the model at different training set sizes. Accuracy improves steadily with more training data, plateauing near full capacity.

$$\text{Syntactic Validity (\%)} = \frac{\text{Number of Valid Outputs}}{\text{Total Outputs}} \times 100$$

Additionally, we categorized failures into two groups: expressions with **unbalanced parentheses** and expressions with **invalid predicate formats**. This helped identify which syntactic features were most frequently violated by the model.

Figure 9 presents the distribution of syntactic categories across 100 test samples. Among these, 72.2% of the outputs were fully valid, 17.1% had unbalanced parentheses, and 10.7% contained malformed predicates.

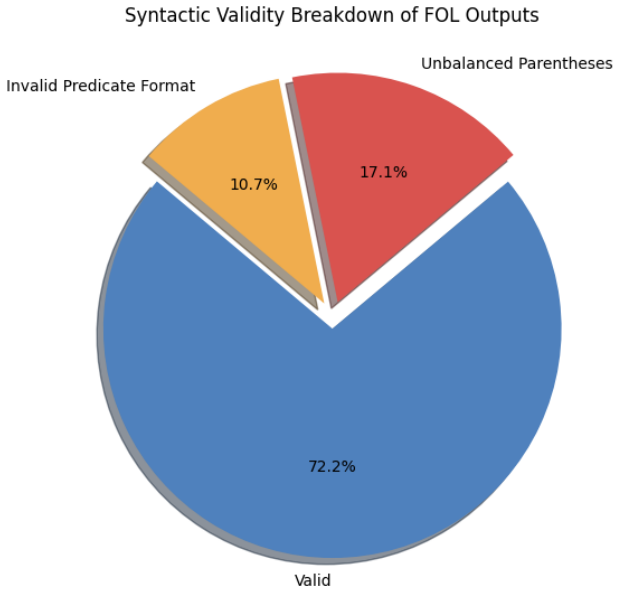

Figure 9: Breakdown of syntactic validity across model-generated FOL expressions. The valid portion satisfies both balanced parentheses and correct predicate format.

8 Discussion

Handling Syntactic Rigor in FOL

To ensure syntactic validity in the generated FOL expressions, we implemented both architectural and evaluation-level safeguards. At the model level, we fine-tuned `phi-2` using prompt-based autoregressive training, allowing it to learn structured output patterns in a token-by-token fashion. During evaluation, we applied rule-based validation checks on parentheses and predicate structure. These post-hoc diagnostics not only quantified syntactic correctness but also revealed specific failure patterns—primarily related to bracket mismatch and malformed function expressions—which future training objectives could target directly.

Managing Quantifier Scope and Nesting

Quantifier scope ambiguity is one of the most difficult aspects of formal logic generation. Our use of a causal language model provided a natural left-to-right decoding structure, which implicitly respects logical nesting and scope boundaries. While the model was not explicitly trained with variable binding rules, the token-level supervision over fully-formed sequences encouraged structural alignment between the input semantics and the FOL output. Accuracy metrics across different FOL lengths showed that the model retained reasonable performance even in long, nested outputs—an indication of learned structural regularity.

Generalization from Limited Data

To combat potential overfitting and data sparsity, we adopted a low-batch, high-accumulation training regime, and conducted controlled experiments on different training set sizes. The learning curve (Figure 8) demonstrated that even small-scale supervision enabled the model to generalize basic logical constructions, while additional training data yielded steady, if diminishing, improvements. This suggests that small LLMs like `phi-2` are well-suited to low-resource symbolic reasoning when paired with consistent token-level training.

Reducing Structural Errors in Long Formulas

One of the major risks in symbolic generation is error accumulation over longer outputs. Our analysis in Figure 7 showed a gradual performance decline with increased formula length—expected due to the compounding complexity. However, the fact that performance plateaued instead of collapsing suggests the model had internalized many syntactic rules and logical motifs. Additionally, we used maximum sequence length (512) and gradient checkpointing to support stable training over long examples without memory exhaustion.

Stable Output Generation and Evaluation

To reduce decoding variability and enforce structural consistency, we used greedy decoding with fixed token limits during inference. This deterministic setting allowed precise evaluation of exact match, BLEU, and F1 scores, and enabled us to isolate true performance trends from sampling noise. By combining generative modeling with symbolic evaluation, our approach balanced flexibility with formal correctness.

9 Future Work

While this study demonstrates that small language models such as `phi-2` can learn to translate natural language into formal logic, several promising directions remain for future research. First, the generation of syntactically valid outputs could be improved by integrating formal constraints directly into the decoding process. This includes constrained beam search or rule-based filters that enforce grammar compliance during generation, thereby reducing invalid logical structures.

Another direction involves moving beyond purely autoregressive modeling. Multi-stage architectures, where one component extracts logical structure and another generates symbolic output, could increase

robustness. Similarly, symbolic-neural hybrid systems may combine the strengths of formal parsers with neural language modeling.

Expanding the task to handle multi-sentence inputs or context-dependent logic generation represents a further challenge. Real-world applications, such as scientific information extraction or legal reasoning, often require logic generation from longer passages, not single-sentence prompts.

Improving dataset diversity is also critical. While MALLS-v0 offers strong baseline coverage, its examples are synthetically constructed and may not reflect the full spectrum of logical expressions seen in practice. Future work could augment the training set with more abstract or naturally occurring examples, using automated tools or weak supervision.

Evaluation techniques should also evolve. Current metrics focus on token-level and structural similarity, but do not assess whether generated formulas are logically equivalent to the ground truth. Incorporating formal logic checkers or theorem provers into the evaluation pipeline could enable more meaningful assessments of correctness.

Finally, scaling to larger pretrained models or exploring few-shot in-context learning for symbolic tasks could further enhance performance. With more compute and richer prompting strategies, it may be possible to achieve stronger generalization with minimal fine-tuning.

In summary, the task of natural language to logic translation remains open-ended, with opportunities to improve generation quality, logical soundness, contextual reasoning, and evaluation rigor.

10 Disclaimer

This report acknowledges the use of AI-based tool ChatGPT[7] for rephrasing certain sections to enhance clarity and readability. Although this project was conducted individually, the pronoun "we" has been used for generality and consistency in academic writing. Additionally, dataset analysis was executed in the free version of Google Colab[8].

References

[1] Jon Barwise. An introduction to first-order logic. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 5–46. Elsevier, 1977.

[2] Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, et al. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*, 2022.

[3] Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. Harnessing the power of large language models for natural language to first-order logic translation. *arXiv preprint arXiv:2305.15541*, 2023.

[4] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.

[5] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[6] Microsoft. Phi-2: Exploring small language models. `https://huggingface.co/microsoft/phi-2`, 2023. Accessed: 2025-04-24.

[7] OpenAI. Chatgpt: A large language model for conversational ai, 2024. Accessed: February 2025.

[8] Google Research. Google colaboratory: Cloud-based python notebook environment, 2024. Accessed: February 2025.