Few-Shot Natural Language to First-Order Logic Translation via Code Generation

Junnan Liu
Department of Data Science & AI, Monash University
to.liujn@outlook.com

Abstract

Translation of natural language to first-order logical formula (NL-FOL) has recently gained significant attention for its critical role in logic-based NLP applications. Some studies attempt to utilize pretrained language models in a sequence-to-sequence manner for the NL-FOL task. However, these methods encounter challenges such as (1) inconsistency between the training and inference phases and (2) the data-intensive and resource-intensive finetuning process. This paper introduces a novel NL-FOL translation method, dubbed **CODE4LOGIC**, which is based on in-context learning and employs code snippets to bridge the gap between natural language and first-order logic. By converting the translation task into a progressive code generation task, CODE4LOGIC demonstrates strong generalization within a training-free manner, and enhances the performance of large language models (LLMs) to generate complex first-order logical formulas. Experimental results on NL-FOL task and downstream task datasets indicate that CODE4LOGIC surpasses prominent training-free baselines and is comparable to supervised models trained on the full training data.

1 Introduction

In recent years, the application of deep neural networks has achieved tremendous success, especially with the emergence of large language models (Ouyang et al., 2022; OpenAI, 2023; Touvron et al., 2023a,b), greatly driving the development of artificial intelligence. Nevertheless, deep neural networks still exhibit limitations, notably in terms of interpretability and comprehensibility. Therefore, the combination of interpretable symbolic logic and machine learning has garnered extensive attention (Dai et al., 2019; Wang et al., 2019). Particularly noteworthy is the task of translating natural language into first-order logical formula (NL-FOL) (Han et al., 2022), which serves as a fundamental component in various logic-based natural language processing (NLP) applications, such as textual entailment (Bos and Markert, 2005), natural language inference (Liu et al., 2021; Yanaka et al., 2019; Suzuki et al., 2019), machine reading comprehension (Liu et al., 2020), and natural language reasoning (Clark et al., 2020; Tafjord et al., 2021; Wang et al., 2022). This challenging task demands a comprehensive understanding of natural language representation, the extraction of essential information, and the subsequent establishment of logical connections among this information (Abzianidze, 2017; Cao et al., 2019).

To tackle this challenge, the community has introduced diverse methods focused on translating natural language to first-order logic. Prior methods rely on handcrafted rules (Bos and Markert, 2005; Zettlemoyer and Collins, 2005; Abzianidze, 2017). As natural language intricacies pose scalability issues, these approaches struggle to extend to practical scenarios. Recently, there has been a growing inclination toward neural methods for addressing this task (Cao et al., 2019; Singh et al., 2020; Hahn et al., 2022). Specifically, some researchers leverage pretrained language models (Devlin et al., 2019; Radford et al., 2018, 2019; Brown et al., 2020; Ouyang et al., 2022; OpenAI, 2023; Touvron et al., 2023a,b) to solve the NL-FOL task in a sequence-to-sequence paradigm (Xu et al., 2024; Yang et al., 2024b; Olausson et al., 2023). Typically, these approaches involve fine-tuning or retraining pretrained language models with logical form data to enhance their efficacy in specific scenarios. Notwithstanding their remarkable performance, they still grapple with several notable challenges: (1) **Inconsistency between training and inference.** Current LLMs are trained via extensive unsupervised pretraining on large-scale general natural language corpora, which lacks first-order logical form data, as with other symbolic data (Wu et al., 2024; Yang et al., 2024b). Conse-

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 10939–10960

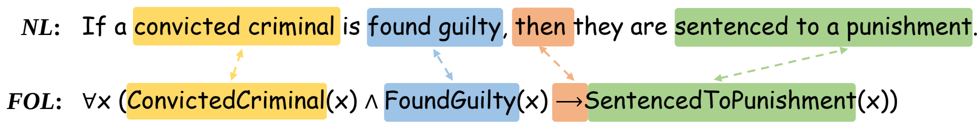

Figure 1: Example of natural language to first-order logic task from the FOILO dataset (Han et al., 2022), which requires the extraction of key information and identification of logical connectives to construct the logical form.

quently, generating the accurate logical form for LLMs poses a considerable challenge (Nie et al., 2024). Moreover, finetuning or retraining large language models with logical form data may lead to catastrophic forgetting in models tailored specifically for natural language tasks (Wu et al., 2024). (2) **Data scarcity and training cost.** Most supervised training methods are data-intensive, requiring a significant number of training examples with expert-annotated steps. Considering both the limited availability of labeled logical form data in practical situations and the computational overhead of finetuning large-scale language models, the generalization capability in the low resource setting of the translation model holds significant importance. Nevertheless, current approaches lack investigating the performance in zero-shot or few-shot scenarios (Shin et al., 2021).

In this paper, we propose an NL-FOL translation method **CODE4LOGIC**, which is based on in-context learning (Brown et al., 2020; Dong et al., 2022) of LLMs. To bridge the gap between natural language and logical form, we utilize pretrained large code models which are pretrained by natural language and programming language (Chen et al., 2021; Li et al., 2023c; Rozière et al., 2023; Gao et al., 2023; Chen et al., 2023) and introduce *code* (*e.g.* Python code snippets) as the intermediary (Wang et al., 2023). Benefiting from the capabilities of LLMs learning from a few examples (Wei et al., 2022a), CODE4LOGIC could demonstrate strong generalization ability within a training-free framework. Specifically, we transform the NL-FOL translation task into a code generation task, enabling us to generate the logical form progressively and enhance the performance of LLMs to generate complex logical formulas (Wei et al., 2022b). Firstly, we define functions corresponding to components of first-order logic and implement them in Python. These functions allow the conversion of each first-order logical formula into a code sequence (*i.e.*, a function call sequence). For a sample of NL-FOL pair, we transform the pair into a code sequence, which, upon execution, yields the first-order logical formula. To translate a query natural language, we provide the defined functions, K demonstration examples in code sequence format, and the query natural language to the LLM. The LLM is then tasked with completing the code sequence corresponding to the query natural language to generate the accurate first-order logic form. Through extensive experiments, we demonstrate that CODE4LOGIC achieves strong generalization performance in the NL-FOL translation task and can also outperform salient baselines on representative downstream logic-based tasks.

We outline our contributions as follows:

- We propose a novel in-context learning method, named CODE4LOGIC, for the NL-FOL translation task. In contrast to current methods, CODE4LOGIC avoids extensive overhead for finetuning and mitigates the reliance on vast training data.
- We convert the NL-FOL translation task to a code generation task to bridge the gap between natural language and the first-order logical formula. By progressively generating the code sequence, our method can boost the generation performance of LLMs when generating complex logical formulas.
- Experimental results on two NL-FOL translation datasets demonstrate that our CODE4LOGIC outperforms salient training-free methods and can achieve comparable results to those fully supervised training methods.
- Moreover, experiments conducted on three categories of downstream tasks validate the availability and efficacy of the first-order logic formulas produced by our method.

2 Related Work

Natural Language to First-Order Logic. The task of translating natural language to first-order logic has long garnered significant attention. Previous researchers attempt to address this problem using a rule-based approach (Bos and Markert, 2005; Zettlemoyer and Collins, 2005; Barker-Plummer et al., 2009; Abzianidze, 2017). Recently, reinforcement learning (Lu et al., 2022) and dual learning (Cao et al., 2019) have been employed to generate first-order logical formulas using neural networks. Drawing inspiration from the machine translation task, some studies have tackled this task using a sequence-to-sequence model (Singh et al., 2020; Xu et al., 2024; Yang et al., 2024b), such as large language models. While large language models exhibit robust generalization capabilities and can handle diverse language structures, they still encounter difficulties in generating accurate logical forms due to their intricate nature. Moreover, certain studies have concentrated on developing datasets (Han et al., 2022; Tian et al., 2021; Yang et al., 2024b), offering substantial assistance for fine-tuning and context-based learning approaches.

In Context Learning with Code-LLMs. In-context learning with large language models (LLMs) (Brown et al., 2020) has exhibited robust few-shot performance across various natural language processing (NLP) tasks, such as question answering (Li et al., 2023d; Nie et al., 2024), information extraction (Pang et al., 2023; Mo et al., 2024), and mathematical reasoning (Lewkowycz et al., 2022; Imani et al., 2023). However, these methodologies encounter challenges in scenarios demanding complex reasoning and handling of symbolic data. Recent studies indicate that LLMs trained with a code corpus exhibit outstanding complex and logical reasoning abilities (Chen et al., 2021; Nijkamp et al., 2022; Gao et al., 2023; Chen et al., 2023), and the programming language is a good bridge between natural language and symbolic language. Consequently, in-context learning with code-LLMs has been applied to tasks requiring complex reasoning, such as knowledge base question answering (Li et al., 2023d; Nie et al., 2024), information extraction (Li et al., 2023b; Wang et al., 2023), and table reasoning (Cheng et al., 2023). Typically, these methodologies transform the task format into code generation, prompting LLMs to accomplish the original task objective through the generation of class instances, supplementary code, or direct generation of required symbolic language (*e.g.*, query SQL).

3 Preliminary

First-Order Logic. First-order logic (FOL) (Barwise, 1977) is a logical system used for reasoning about the properties of objects. It involves quantified variables over non-logical objects, allowing the formation of sentences with variables. This structure facilitates statements like *there exists x such that x is Socrates and x is a man*, which differs from simple propositions such as *Socrates is a man*. In this context, there exists functions as a quantifier, with x representing a variable. This logic consists of two key components: syntax governs the construction of valid symbol sequences in first-order logic, while semantics clarifies the interpretations of these expressions. In this paper, we include the basic elements of first-order logic as follows:

- **Constants.** Constants represent individuals in the world, such as `Boy`, `Socrates`.
- **Variables.** Variables indicate variable symbol, such as x, y, z.
- **Function.** Function maps individuals to individuals, for example, `FatherOf(Tom)` means the father of Tom.
- **Predicate.** Predicate maps individuals to truth values, such as `Greater`(x, y).
- **Logical Connectives.** Logical connectives include $\wedge, \vee, \oplus, \rightarrow, \leftrightarrow$.
- **Quantifier Symbols.** Quantifier symbols consist of $\forall$ for universal quantification, and $\exists$ for existential quantification.
- **Equality Symbols.** Equality symbols can be divided into equivalence $=$ and non-equivalence $\neq$.
- **Punctuation Symbols.** Punctuation symbols include brackets, dots, and etc.

The full Backus-Naur Form grammar to induce any first-order logical formula can be found in Appendix A.

4 Method

In this section, we will introduce how to translate the natural language to the first-order logic via CODE4LOGIC. Initially, the first-order logical formula is parsed into the tree structure (Section 4.2). Each node in the tree can be viewed as a function that generates a subformula of the first-order logical formula, which can be easily implemented in Python. Based on this, we can construct a code sequence given a first-order logical formula (Section 4.2). Subsequently, when a new natural language input necessitates translation, an LLM is employed to create a code sequence of function calls based on demonstration examples (Section 4.3). Finally, a program interpreter is utilized to execute the generated code sequence and derive the first-order logical formula. We also discuss the implementation details and the utilization of resulting first-order logical formulas for downstream tasks in Appendix D.1.

4.1 Task Formulation

Given a natural language statement x_{nl}, the NL-FOL task aims to translate it to the first-order logical formula x_{fol}. In this paper, we reformulate the task as a code generation task, where x_{nl} and x_{fol} are denoted in code form as x^c_{nl} and x^c_{fol}, respectively. When the translation model (e.g., LLM) is provided with x^c_{nl} as input, its objective is to generate x^c_{fol}. Within the context of in-context learning, the translation model receives a limited set of annotated data comprising K demonstration examples in the format of pairs $\left\{\left(x^c_{nl,i}, x^c_{fol,i}\right)\right\}_{i=1}^{K}$.

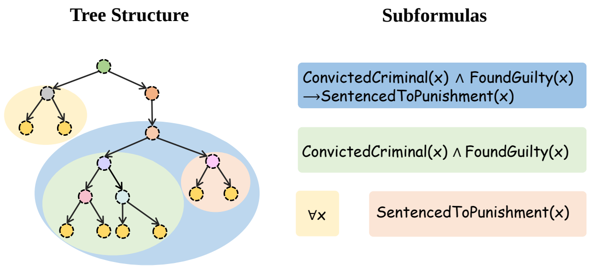

Figure 2: The tree structure of the first-order logical formula $\forall x(\texttt{ConvictedCriminal}(x) \land \texttt{FoundGuilty}(x) \rightarrow \texttt{SentencedToPunishment}(x))$.

4.2 Parse First-Order Logic to Code Sequence

To perform in-context learning for NL-FOL translation in the code generation manner, the primary challenge lies in the conversion of a first-order logical formula into a code sequence. In practice, we parse the first-order logical formula to a tree structure according to the BNF (Backus Normal Form) grammar. The hierarchical tree structure offers an effective approach for obtaining code sequence: by conceptualizing each node in the tree as a function that processes the child node's formula and produces the formula representation of the current substructure, we can systematically reconstruct the original first-order logical formula in a bottom-up fashion. For instance, consider the following first-order logical formula:

$$\forall x\,(\texttt{ConvictedCriminal}(x) \land \texttt{FoundGuilty}(x) \rightarrow \texttt{SentencedToPunishment}(x))\,.$$

Figure 2 shows the corresponding tree structure. Moreover, this procedure can be articulated directly using a programming language format. We can use the pseudo-code sequence in Algorithm 1 to generate the first-order logical formula in Figure 2. In light of the components of first-order logic, we have devised 15 fundamental functions for generating first-order logical formulas. We opt for Python as the implementation platform for these functions, building upon the proven success of Code-LLMs (Chen et al., 2021; Li et al., 2023c) in Python. For comprehensive information on the implementation of all foundational functions, please refer to Appendix B.

Algorithm 1 Pseudo code sequence for the first-order logical formula $\forall x(\texttt{ConvictedCriminal}(x) \land \texttt{FoundGuilty}(x) \rightarrow \texttt{SentencedToPunishment}(x))$.

```
formula_x = Variable('x')
formula1 = Predicate('Convictedcriminal', formula_x)
formula2 = Predicate('Foundguilty', formula_x)
formula3 = Predicate('Sentencedtopunishment',
                    formula_x)
formula = Conjunction(formula1, formula2)
formula = Implication(formula, formula3)
formula = UniversalQuantification(formula,
                                   formula_x)
```

Also, we provide the pseudo-code for how to convert the FOL tree structure to Python code sequence in Appendix C.

4.3 In-Context Learning for NL-FOL Translation

In the realm of in-context learning (Brown et al., 2020), a method typically involves a set of demonstration examples $\{e_i\}_{i=1}^{K}$ and a query q. Certain tasks may necessitate an additional task description or instruction denoted as p. Then, a large language model is tasked with generating the output y defined as:

$$y = \text{LLM}\left(p; \{e_i\}_{i=1}^{K}; q\right). \quad (1)$$

Here, y may take the form of a label in classification tasks (Shome and Yadav, 2023; Yang et al., 2024a) or a sequence in generative tasks (Li et al., 2023a; Mathur et al., 2023; Tang et al., 2023). Figure 3 outlines the framework of CODE4LOGIC and the aforementioned components, integrated within the framework of CODE4LOGIC, will be further elaborated upon in the subsequent paragraphs.

Task Description. Task description serves to offer detailed guidance to the model regarding the

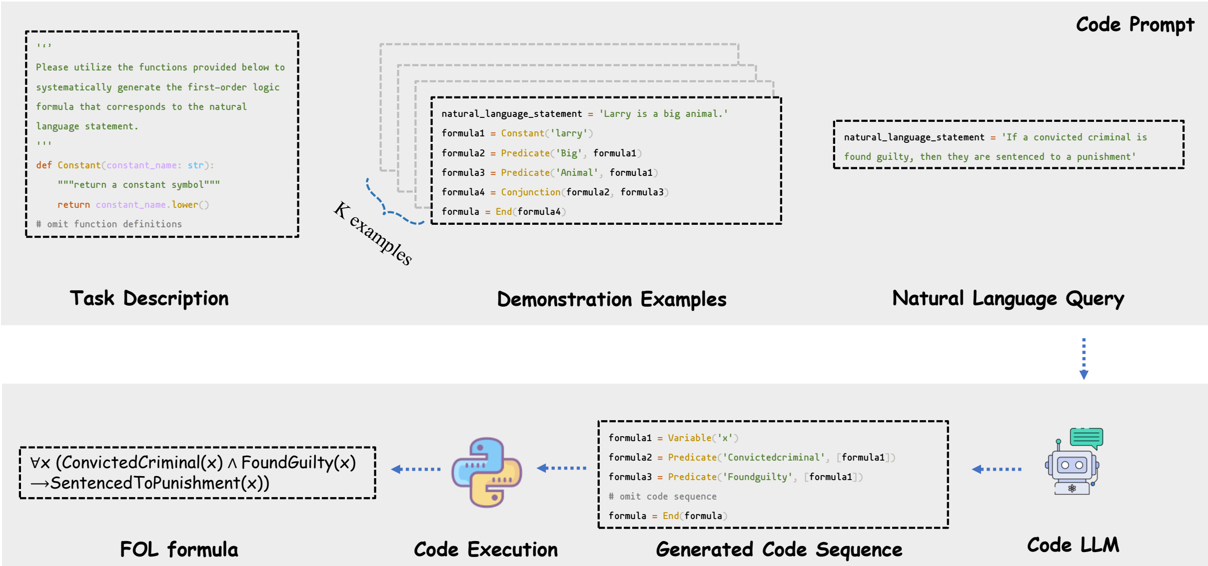

Figure 3: Overall process of CODE4LOGIC.

```
'''
Please utilize the functions provided below
to systematically generate the first-order
logic formula that corresponds to the nat-
ural language statement.
'''
```

Figure 4: Python code for comment-style instruction.

task. In CODE4LOGIC, the task description is segmented into two components: (1) comment-style instructions and (2) implementation of basis functions similar to previous work (Wang et al., 2023; Li et al., 2023b). As previously stated, all basis functions are implemented in Python, thus the instructions are also provided in the form of Python comments, as demonstrated in Figure 4. Subsequently, the comment-style instructions and the implementation of basis functions outlined in Appendix B are merged to compose the task description p.

Demonstration Examples. Demonstration examples consist of a series of NL-FOL code pairs $\left\{\left(x_{nl,i}^{c}, x_{fol,i}^{c}\right)\right\}_{i=1}^{K}$. Each demonstration example consists of (1) a natural language statement in Python and (2) a code sequence to generate the corresponding first-order logical formula. The natural language statement is transformed into a Python assignment statement, where the natural language string is assigned to the variable `natural_language_statement`. The code sequence is obtained from the parse tree of the first-order logical formula mentioned in Section 4.2. Referring to the first-order logical formula in Figure 1 and Figure 2, the demonstration example will be reformulated as shown in Figure 5. Following the execution of the aforementioned

```
natural_language_statement = 'If a convicted criminal is ' + \
'found guilty, then they are sentenced to a punishment.'
formula1 = Variable('x')
formula2 = Predicate('Convictedcriminal ', [formula1])
formula3 = Predicate('Foundguilty ', [formula1])
formula4 = Predicate('Sentencedtopunishment ', [formula1])
formula5 = Conjunction(formula2, formula3)
formula6 = Implication(formula5, formula4)
formula8 = UniversalQuantification(formula6, formula1)
formula = End(formula8)
```

Figure 5: Code sequence for *If a convicted criminal is found guilty, then they are sentenced to a punishment.*

Python code sequence, the first-order logical formula can be retrieved by accessing the `formula` variable. To generate the complete set of demonstration examples, K pairs of (x_{nl}, x_{fol}) are randomly sampled from the support dataset and transformed into $\left\{\left(x_{nl,i}^{c}, x_{fol,i}^{c}\right)\right\}_{i=1}^{K}$. Then, these pairs are concatenated into a unified prompt denoted by $\left(x_{nl,1}^{c}, x_{fol,1}^{c}\right) \oplus \left(x_{nl,2}^{c}, x_{fol,2}^{c}\right) \ldots \oplus \left(x_{nl,K}^{c}, x_{fol,K}^{c}\right)$. Here, $\oplus$ denotes the concatenation function.

Natural Language Query. Recall that our target is to generate the code sequence for retrieving the first-order logical formula for a new query. Thus, we transform a new natural language statement into a Python assignment statement, aligning it with the structure defined in the demonstration examples. The LLM-based translation model is prompted to complete the following code sequence corresponding to the new natural language statement via in-context learning.

5 Experiments

5.1 Experiments on NL-FOL Translation

5.1.1 Setup

Datasets. We use two mainstream datasets for the NL-FOL translation task, FOLIO (Han et al., 2022) and MALLS (Yang et al., 2024b). Specifically, we use the validation set of FOLIO and the test set of MALLS to assess the performance of CODE4LOGIC and other baselines.

Baselines. We mainly compare our model with two salient supervised training methods LOGICLLAMA (Yang et al., 2024b) and Symbol-LLM (Xu et al., 2024). Both of these models stem from the fine-tuning or retraining of large language models using NL-FOL translation datasets. Additionally, we present performance results for Flan-T5 (Raffel et al., 2020; Chung et al., 2022) and Claude-1 (Perez et al., 2023), CodeGeeX2 (Zheng et al., 2023a), GPT-3.5 (Ouyang et al., 2022), and GPT-4 (OpenAI, 2023) under few-shot settings, which is based on the text prompt. Please refer to Appendix D.3.2 for more details about baselines.

Metrics. Common translation and generation tasks often employ Rouge (Lin, 2004) and Bleu (Papineni et al., 2002) as metrics, evaluating n-gram overlap between the reference and candidate. Nevertheless, these metrics are inadequate for NL-FOL translation tasks since the focus is on logical equivalence rather than exact word-level matches. Aligning with previous works (Yang et al., 2024b; Xu et al., 2024), we utilize Logic Equivalence (LE) as the evaluation metric. LE is determined by comparing the truth tables of the reference first-order logical formula and the candidate version, followed by calculating the overlap ratio. For further details, please refer to the original paper (Yang et al., 2024b).

5.1.2 Results

CODE4LOGIC v.s. Supervised Training Methods. As shown in Table 1, CODE4LOGIC surpasses LOGICLLAMA, showcasing strong capabilities in the NL-FOL translation task. Specifically, CODE4LOGIC $_{\texttt{gpt-3.5-turbo-16k}}$ leads to improvements of 7.77 and 9.58 over LOGICLLAMA for the FOLIO and MALLS datasets, respectively. Moreover, despite operating in a training-free manner, CODE4LOGIC demonstrates surpassing performance relative to Symbol-LLM, which is trained on a more extensive symbolic dataset than LOGICLLAMA.

CODE4LOGIC v.s. Text Prompt Based Methods. To illustrate the effectiveness of using code prompts, we proceed to compare CODE4LOGIC with a text prompt-based baseline under the same LLM framework. The results presented in Table 1 indicate that employing a code prompt significantly enhances performance across both datasets, emphasizing the advantages of utilizing code as a bridge between natural and symbolic languages.

Performance w.r.t. Different Code-LLMs. We compare the performance of GPT-3.5 and CodeGeeX to measure the influence of different Code-LLMs. Notably, GPT-3.5 outperforms CodeGeeX due to its superior text understanding capabilities. CodeGeeX, on the other hand, having a smaller parameter count (6b), is hindered by its lack of alignment as a code completion model.

Performance w.r.t. Different Number of Demonstration Examples. As shown in Figure 6, we conduct the experiment concerning different numbers of demonstration examples. The model's performance improves with an increased number of demonstration examples. However, the performance gains become marginal once a certain threshold is reached. Notably, even with a limited number of demonstration examples, CODE4LOGIC delivers competitive results.

Ablation Study. To assess the significance of each component of the prompt in CODE4LOGIC, we conduct ablation experiments as detailed in Table 2. The findings indicate that excluding the *comment-style instruction*, *basis function definition*, or *demonstration examples* led to a significant decline in model performance. Specifically, the absence of the *comment-style instruction* could obscure the task definition, resulting in incorrect model outputs. Similarly, omitting the *basis function definition* hinders the ability of Code-LLMs to comprehend essential functions, thus impeding the generation of code sequences. The removal of *demonstration examples* complicates the Code-LLMs' understanding of task requirements solely based on instructions. Additionally, we switch components from text-based prompts to evaluate the efficacy of code-based prompts. Analysis of the experimental outcomes reveals a slight performance decline with text-based Comment-Style Instructions, while a notable performance drop occurs

Model	Prompt Type	FOLIO LE	MALLS LE
Flan-T5 (Chung et al., 2022)	text	$70.67_{\pm0.27}$	$68.45_{\pm0.35}$
Claude-1 (Perez et al., 2023)	text	$74.47_{\pm0.69}$	$77.46_{\pm0.81}$
GPT4 (OpenAI, 2023)	text	$85.53_{\pm0.15}$	$84.38_{\pm0.34}$
LogicLLaMA (Yang et al., 2024b)	text	84.90	81.34
Symbol-LLM-7b$_{\texttt{single_sft}}$ (Xu et al., 2024)	text	90.81	89.24
Symbol-LLM-7b (Xu et al., 2024)	text	90.58	88.88
Symbol-LLM-13b$_{\texttt{single_sft}}$ (Xu et al., 2024)	text	$91.59^{\star}$	89.41
Symbol-LLM-13b (Xu et al., 2024)	text	90.65	$89.50^{\star}$
CodeGeeX2 (Zheng et al., 2023a)	text	$56.71_{\pm0.42}$	$57.83_{\pm0.21}$
GPT3.5 (Ouyang et al., 2022)	text	$83.53_{\pm0.15}$	$82.72_{\pm0.33}$
Code4Logic $_{\text{CodeGeeX}}$	code	$84.77_{\pm0.05}$	$85.81_{\pm0.02}$
Code4Logic $_{\texttt{gpt3.5-turbo-16k}}$	code	$\mathbf{92.67}_{\pm0.03}$	$\mathbf{90.92}_{\pm0.04}$

Table 1: Experimental results on FOLIO and MALLS datasets. Our approach is indicated by the use of light grey shading. The results for Flan-T5, Claude-1, and GPT-3.5 were obtained by prompting these models in a few-shot setting. Additionally, the text prompt based results of GPT-3.5 and CodeGeeX are reported for a clear comparative analysis. $\star$ means the best results of the supervised training methods.

Model	FOLIO LE	MALLS LE
CodeGeeX		
Code4Logic	84.77	85.81
w/o Comment-Style Instruction	30.45	31.77
w Text-based Comment-Style Instruction	82.68	83.71
w/o Basis Function Definition	21.78	20.29
w Text-based Basis Function Definition	20.58	22.81
w/o Demonstration Examples	24.61	22.78
w Text-based Demonstration Examples	27.64	23.11
GPT-3.5		
Code4Logic	92.67	90.92
w/o Comment-Style Instruction	29.33	29.61
w/o Basis Function Definition	22.82	23.47
w/o Demonstration Examples	24.91	25.19

Table 2: Ablation Study on FOLIO and MALLS. We only report the mean of experimental results.

with text-based Basis Function Definitions and text-based Demonstration Examples. While text-based Comment-Style Instructions, resembling NL-based prompts, yield outcomes akin to Python comment-style instructions, text-based Basis Function Definitions and Demonstration Examples impede effective information provision due to their incongruity with code-style prompts.

5.2 Experiments on Downstream Tasks

First-order logical formulas derived from natural language expressions enable the completion of various downstream tasks, such as textual entailment (Bos and Markert, 2005) and natural language reasoning (Clark et al., 2020; Tafjord et al., 2021; Wang et al., 2022). These tasks are commonly tackled through logical reasoning or proving using external tools like Prover9 (McCune, 2005–2010)[1] and Pyke (Frederiksen, 2008)[2].

[1] https://www.cs.unm.edu/~mccune/prover9/
[2] https://pyke.sourceforge.net/

To verify the validity of Code4Logic in downstream tasks, we generate the code sequence using the proposed framework and subsequently execute Python code to acquire the first-order logic form, which is readily convertible and processable by external logical reasoning engines. Finally, we employ an external logical reasoning engine to derive the output. A similar process can be found in previous work (Pan et al., 2023).

However, there may be ambiguity errors due to the randomness of the generated results of large language models. For instance, the terms `UK` and `UnitedKingdom` may both refer to England, yet they are treated as distinct entities by the reasoning engine. To solve this problem, we first tokenize all the first-order logical formulas into tokens. Subsequently, we employ a merging algorithm based on semantic similarity to unify the statements that may express the same concept. Finally, the unified first-order logical formulas are fed into the reasoning engine for following computational processing.

5.2.1 Setup

Datasets. We conduct experiments on three types of downstream tasks: natural language inference, logical reasoning, and fact-checking. We chose three representative datasets: LogicNLI (Tian et al., 2021) for natural language inference, RuleTaker (Clark et al., 2020) for logical reasoning, and VitaminC (Schuster et al., 2021) for fact-checking.

- **LogicNLI.** LogicNLI is a NLI dataset. LogicNLI effectively separates the targeted FOL reasoning from common-sense inference and comprises 16k, 2k, and 2k samples for the training, validation, and test sets, respectively.

Model	LogicNLI Acc
Flan-T5	$70.11_{\pm0.16}$
LLaMA	$75.87_{\pm0.21}$
GPT-3.5	$82.44_{\pm0.11}$
BERT	$55.92_{\pm0.05}$
RoBERTa	$68.37_{\pm0.02}^{\star}$
XLNet	$65.41_{\pm0.02}$
CODE4LOGIC CodeGeeX	$85.78_{\pm0.24}$
CODE4LOGIC gpt3.5-turbo	$\mathbf{91.34}_{\pm0.28}$

Table 3: results on LogicNLI.

Model	RuleTaker Acc
Flan-T5	$51.26_{\pm0.12}$
LLaMA	$48.89_{\pm0.09}$
GPT-3.5	$54.74_{\pm0.14}$
RoBRTa	$53.50_{\pm0.03}$
Neural Unifier	$59.73_{\pm0.12}^{\star}$
CODE4LOGIC CodeGeeX	$56.36_{\pm0.19}$
CODE4LOGIC gpt3.5-turbo	$\mathbf{61.43}_{\pm0.21}$

Table 4: results on RuleTaker.

Model	VitaminC Acc
Flan-T5	$55.33_{\pm0.05}$
LLaMA	$57.47_{\pm0.14}$
GPT-3.5	$61.26_{\pm0.33}$
FactCC	$54.71_{\pm0.01}$
BLANC	$55.73_{\pm0.04}$
BARTSCORE	$64.22_{\pm0.02}^{\star}$
CODE4LOGIC CodeGeeX	$55.61_{\pm0.21}$
CODE4LOGIC gpt3.5-turbo	$\mathbf{68.39}_{\pm0.13}$

Table 5: Results on VitaminC.

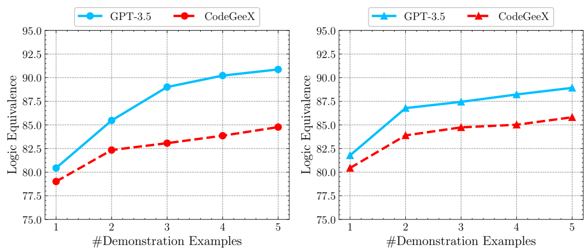

Figure 6: Performance of the different number of demonstration examples. The left part is the results of FOLIO and the right part is the results of MALLS.

- **RuleTaker.** RuleTaker is a large-scale dataset, which is designed for logical reasoning tasks. RuleTaker consists of 480k, 75.9k, and 152k samples for the training, validation, and test sets, respectively.
- **VitaminC.** VitaminC contains more than 450k claim-evidence pairs for fact-checking based on over 450k claim-evidence pairs sourced from over 100k revisions of popular Wikipedia pages, including “synthetic” revisions.

Baselines. We mainly evaluate CODE4LOGIC against representative large language models (LLMs), such as Flan-T5 (Raffel et al., 2020; Chung et al., 2022), LLaMA (Touvron et al., 2023a), and GPT-3.5 (Ouyang et al., 2022), which are under few-shot settings. Additionally, we compare it with some prominent baseline methods (Picco et al., 2021; Kryscinski et al., 2020; Vasilyev et al., 2020; Yuan et al., 2021) that have been trained on the full training set. Please refer to Appendix D.4.2 for more details about baselines.

Metrics. We approach the three task types as multi-classification tasks, with the average accuracy serving as the performance metric. CODE4LOGIC and other few-shot methods are evaluated directly on the test set. In contrast, additional methods are trained on the training set, showcasing the best performance on the test set chosen using the validation set.

```
natural_language_statement = ('If something'
' is green and cold then it is furry.')
formula1 = Variable('x')
formula2 = Predicate('Green', [formula1])
formula3 = Predicate('Cold', [formula1])
formula4 = Predicate('Furry', [formula1])
formula5 = Conjunction(formula2, formula3)
formula6 = Implication(formula5, formula4)
formula7 = UniversalQuantification(
           formula6, formula1)
formula = End(formula7)
```

$\forall x(\texttt{Green}(x) \land \texttt{Cold}(x) \rightarrow \texttt{Furry}(x))$

Figure 7: A case of the code sequence generated by CODE4LOGIC.

5.2.2 Results

Performance on LogicNLI. The results for LogicNLI are presented in Table 3. CODE4LOGIC significantly outperforms fully supervised training baselines. This superiority can be attributed to the specific requirement of logical reasoning imposed by the LogicNLI dataset, a criterion for which previous pre-trained models (*e.g.*, BERT, and RoBERTa) exhibit limitations in their logical reasoning capabilities.

Performance on RuleTaker. RuleTaker dataset requires models to have complex logical reasoning ability compared to LogicNLI. The performance of CODE4LOGIC and other baselines is shown in Table 4. We can find that CODE4LOGIC surpasses salient baselines, which demonstrates the effectiveness of our CODE4LOGIC to understand complex logical structures.

Performance on VitaminC. The experimental results on the VitaminC dataset are presented in Table 5. The fact-checking task requires the model to have a more complete understanding of the text structure and the facts contained within it than the previous task. We can find that our CODE4LOGIC outperforms the few-shot methods reliant on large language models, indicating the efficacy of employing logical expressions as transitional tools for addressing the fact-checking task. Furthermore, our CODE4LOGIC exhibits comparable performance over conventional text generation evaluation models, providing additional confirmation of our hypothesis.

Case Study. Additionally, we illustrate several cases of the code sequence generated by CODE4LOGIC and the corresponding first-order logical formula, which are available in Figure 7 and Appendix E.2. These cases demonstrate CODE4LOGIC's capability to produce efficient code sequences and high-quality first-order logical formulas.

6 Conclusion

This paper introduces **CODE4LOGIC**, a training-free method that leverages the code based prompt for NL-FOL translation task within an in-context learning framework. Through utilizing code as a connector linking natural language and first-order logical formulas, our approach adeptly bridges the gap between training and inference in large language models, showcasing robust generalization capabilities. Extensive experiments demonstrate the superiority of our model not only in NL-FOL translation but also in various downstream tasks.

Limitations

In this paper, we introduce a new NL-FOL translation method. We believe this method still has much room for improvement:

- Multilingual expansion. Currently, our method is constrained to translation between English and first-order logical formulas, necessitating additional investigation for other languages;
- Inference efficiency. Since our approach is based on a large language model and a long prompt, our approach is less efficient when inference, which is sensitive in some scenarios.

References

Lasha Abzianidze. 2017. Langpro: Natural language theorem prover. In *EMNLP (System Demonstrations)*, pages 115–120. Association for Computational Linguistics.

Priyanka Agrawal, Ayushi Dalmia, Parag Jain, Abhishek Bansal, Ashish R. Mittal, and Karthik Sankaranarayanan. 2019. Unified semantic parsing with weak supervision. In *ACL*, pages 4801–4810. Association for Computational Linguistics.

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *ACL*, pages 47–52. The Association for Computer Linguistics.

Dave Barker-Plummer, Richard Cox, and Robert Dale. 2009. Dimensions of difficulty in translating natural language into first-order logic. In *EDM*, pages 220–229.

Jon Barwise. 1977. An introduction to first-order logic. In *Stud. Logic Found*, volume 90, pages 5–46. Elsevier.

Johan Bos and Katja Markert. 2005. Recognising textual entailment with logical inference. In *HLT/EMNLP*, pages 628–635. The Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *NeurIPS*.

Ruisheng Cao, Su Zhu, Chen Liu, Jieyu Li, and Kai Yu. 2019. Semantic parsing with dual learning. In *ACL*, pages 51–64. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. 2023. Binding language models in symbolic languages. In *ICLR*. OpenReview.net.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. In *IJCAI*, pages 3882–3890. ijcai.org.

Wang-Zhou Dai, Qiu-Ling Xu, Yang Yu, and Zhi-Hua Zhou. 2019. Bridging machine learning and logical reasoning by abductive learning. In *NeurIPS*, pages 2811–2822.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186. Association for Computational Linguistics.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: general language model pretraining with autoregressive blank infilling. In *ACL*, pages 320–335. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Bruce Frederiksen. 2008. Applying expert system technology to code reuse with pyke. *PyCon: Chicago*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: program-aided language models. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Christopher Hahn, Frederik Schmitt, Julia J. Tillman, Niklas Metzger, Julian Siber, and Bernd Finkbeiner. 2022. Formal specifications from natural language. *CoRR*, abs/2206.01962.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq R. Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. 2022. FOLIO: natural language reasoning with first-order logic. *CoRR*, abs/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 *ICLR*. OpenReview.net.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models. In *ACL (industry)*, pages 37–42. Association for Computational Linguistics.

Wojciech Kryscinski, Bryan McCann, Caiming Xiong, and Richard Socher. 2020. Evaluating the factual consistency of abstractive text summarization. In *EMNLP*, pages 9332–9346. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020.

BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *ACL*, pages 7871–7880. Association for Computational Linguistics.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay V. Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. In *NeurIPS*.

Chunyou Li, Mingtong Liu, Hongxiao Zhang, Yufeng Chen, Jinan Xu, and Ming Zhou. 2023a. MT2: towards a multi-task machine translation model with translation-specific in-context learning. In *EMNLP*, pages 8616–8627. Association for Computational Linguistics.

Peng Li, Tianxiang Sun, Qiong Tang, Hang Yan, Yuanbin Wu, Xuanjing Huang, and Xipeng Qiu. 2023b. Codeie: Large code generation models are better few-shot information extractors. In *ACL*, pages 15339–15353. Association for Computational Linguistics.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Moustafa-Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023c. Starcoder: may the source be with you! *CoRR*, abs/2305.06161.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023d. Few-shot in-context learning on knowledge base question answering. In *ACL*, pages 6966–6980. Association for Computational Linguistics.

Percy Liang. 2013. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Hanmeng Liu, Leyang Cui, Jian Liu, and Yue Zhang. 2021. Natural language inference in context - investigating contextual reasoning over long texts. In *AAAI*, pages 13388–13396. AAAI Press.

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. In *IJCAI*, pages 3622–3628. ijcai.org.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Xuantao Lu, Jingping Liu, Zhouhong Gu, Hanwen Tong, Chenhao Xie, Junyang Huang, Yanghua Xiao, and Wenguang Wang. 2022. Parsing natural language into propositional and first-order logic with dual reinforcement learning. In *COLING*, pages 5419–5431. International Committee on Computational Linguistics.

Yash Mathur, Sanketh Rangreji, Raghav Kapoor, Medha Palavalli, Amanda Bertsch, and Matthew R. Gormley. 2023. Summqa at mediqa-chat 2023: In-context learning with GPT-4 for medical summarization. In *ClinicalNLP@ACL*, pages 490–502. Association for Computational Linguistics.

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

Ying Mo, Jian Yang, Jiahao Liu, Shun Zhang, Jingang Wang, and Zhoujun Li. 2024. C-ICL: contrastive in-context learning for information extraction. *CoRR*, abs/2402.11254.

Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. 2024. Code-style in-context learning for knowledge-based question answering. In *AAAI*, pages 18833–18841. AAAI Press.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A conversational paradigm for program synthesis. *CoRR*, abs/2203.13474.

Theo 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. In *EMNLP*, pages 5153–5176. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke

Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

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

Chaoxu Pang, Yixuan Cao, Qiang Ding, and Ping Luo. 2023. Guideline learning for in-context information extraction. In *EMNLP*, pages 15372–15389. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318. ACL.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035.

Ethan Perez, Sam Ringer, Kamile Lukosiute, Karina Nguyen, Edwin Chen, Scott Heiner, Craig Pettit, Catherine Olsson, Sandipan Kundu, Saurav Kadavath, Andy Jones, Anna Chen, Benjamin Mann, Brian Israel, Bryan Seethor, Cameron McKinnon, Christopher Olah, Da Yan, Daniela Amodei, Dario Amodei, Dawn Drain, Dustin Li, Eli Tran-Johnson, Guro Khundadze, Jackson Kernion, James Landis, Jamie Kerr, Jared Mueller, Jeeyoon Hyun, Joshua Landau, Kamal Ndousse, Landon Goldberg, Liane Lovitt, Martin Lucas, Michael Sellitto, Miranda Zhang, Neerav Kingsland, Nelson Elhage, Nicholas Joseph, Noemí Mercado, Nova DasSarma, Oliver Rausch, Robin Larson, Sam McCandlish, Scott Johnston, Shauna Kravec, Sheer El Showk, Tamera Lanham, Timothy Telleen-Lawton, Tom Brown, Tom Henighan, Tristan Hume, Yuntao Bai, Zac Hatfield-Dodds, Jack Clark, Samuel R. Bowman, Amanda Askell, Roger Grosse, Danny Hernandez, Deep Ganguli, Evan Hubinger, Nicholas Schiefer, and Jared Kaplan. 2023. Discovering language model behaviors with model-written evaluations. In *ACL (Findings)*, pages 13387–13434. Association for Computational Linguistics.

Gabriele Picco, Thanh Lam Hoang, Marco Luca Sbodio, and Vanessa López. 2021. Neural unification for logic reasoning over natural language. In *EMNLP (Findings)*, pages 3939–3950. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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. *JMLR*, 21:140:1–140:67.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, 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. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Tal Schuster, Adam Fisch, and Regina Barzilay. 2021. Get your vitamin c! robust fact verification with contrastive evidence. In *NAACL-HLT*, pages 624–643. Association for Computational Linguistics.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *EMNLP*, pages 7699–7715. Association for Computational Linguistics.

Debaditya Shome and Kuldeep Yadav. 2023. Exnet: Efficient in-context learning for data-less text classification. *CoRR*, abs/2305.14622.

Hrituraj Singh, Milan Aggarwal, and Balaji Krishnamurthy. 2020. Exploring neural models for parsing natural language into first-order logic. *CoRR*, abs/2002.06544.

Riko Suzuki, Hitomi Yanaka, Masashi Yoshikawa, Koji Mineshima, and Daisuke Bekki. 2019. Multimodal logical inference system for visual-textual entailment. In *ACL*, pages 386–392. Association for Computational Linguistics.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. In *ACL/IJCNLP (Findings)*, Findings of ACL, pages 3621–3634. Association for Computational Linguistics.

Xiangru Tang, Andrew Tran, Jeffrey Tan, and Mark Gerstein. 2023. Gersteinlab at mediqa-chat 2023: Clinical note summarization from doctor-patient conversations through fine-tuning and in-context learning. In *ClinicalNLP@ACL*, pages 546–554. Association for Computational Linguistics.

Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. Diagnosing the first-order logical reasoning ability through logicnli. In

EMNLP, pages 3738–3747. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Oleg V. Vasilyev, Vedant Dharnidharka, and John Bohannon. 2020. Fill in the BLANC: human-free quality estimation of document summaries. In *Eval4NLP*, pages 11–20. Association for Computational Linguistics.

Po-Wei Wang, Priya L. Donti, Bryan Wilder, and J. Zico Kolter. 2019. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, volume 97, pages 6545–6554. PMLR.

Siyuan Wang, Wanjun Zhong, Duyu Tang, Zhongyu Wei, Zhihao Fan, Daxin Jiang, Ming Zhou, and Nan Duan. 2022. Logic-driven context extension and data augmentation for logical reasoning of text. In *ACL (Findings)*, pages 1619–1629. Association for Computational Linguistics.

Xingyao Wang, Sha Li, and Heng Ji. 2023. Code4struct: Code generation for few-shot event structure prediction. In *ACL*, pages 3640–3663. Association for Computational Linguistics.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. Emergent abilities of large language models. *TMLR*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022b. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ping Luo, and Ying Shan. 2024. Llama pro: Progressive llama with block expansion. *CoRR*, abs/2401.02415.

Fangzhi Xu, Zhiyong Wu, Qiushi Sun, Siyu Ren, Fei Yuan, Shuai Yuan, Qika Lin, Yu Qiao, and Jun Liu. 2024. Symbol-llm: Towards foundational symbol-centric interface for large language models. In *ACL*, pages 13091–13116. Association for Computational Linguistics.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. 2019. HELP: A dataset for identifying shortcomings of neural models in monotonicity reasoning. In *SEM@NAACL-HLT*, pages 250–255. Association for Computational Linguistics.

Songhua Yang, Xinke Jiang, Hanjie Zhao, Wenxuan Zeng, Hongde Liu, and Yuxiang Jia. 2024a. Faima: Feature-aware in-context learning for multi-domain aspect-based sentiment analysis. *CoRR*, abs/2403.01063.

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. 2024b. Harnessing the power of large language models for natural language to first-order logic translation. In *ACL*, pages 6942–6959. Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, pages 5754–5764.

Weizhe Yuan, Graham Neubig, and Pengfei Liu. 2021. Bartscore: Evaluating generated text as text generation. In *NeurIPS*, pages 27263–27277.

Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. 2023. GLM-130B: an open bilingual pre-trained model. In *ICLR*. OpenReview.net.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI*, pages 658–666. AUAI Press.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023a. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *KDD*, pages 5673–5684. ACM.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Zihan Wang, Lei Shen, Andi Wang, Yang Li, Teng Su, Zhilin Yang, and Jie Tang. 2023b. Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x. *CoRR*, abs/2303.17568.

A BNF Grammar of First-Order Logic

The BNF grammar of first-order logic is illustrated in Figure 8.

B Implementation of Basis Functions

The definitions of basis functions in Python are as follows:

```
def Constant(constant_name: str):
    """return a constant symbol"""
    return constant_name.lower()


def Variable(variable_name: str):
    """return a variable symbol,
    which starts with `$`"""
    return '$' + variable_name


def Function(function_name: str, terms: List[str]):
    """return a function symbol, for example,
    father(x) means the father of x"""
    return '{}({})'.format(
        function_name.lower(), ', '.join(terms))


def Predicate(predicate_name: str, terms: List[str]):
    """return an atomic formula with a predicate,
    whose name starts with uppercase"""
    return '{}({})'.format(
        predicate_name.lower().capitalize(), ', '.join(terms))


def Equal(term_a: str, term_b: str):
    """return an atomic formula with equal operation"""
    return '{} = {}'.format(term_a, term_b)


def NonEqual(term_a: str, term_b: str):
    """return an atomic formula with equal operation"""
    return '{} \u2260 {}'.format(term_a, term_b)


def Negation(formula: str):
    """return the negation of the input formula"""
    return '\u00ac({})'.format(formula)


def Conjunction(formula_a: str, formula_b: str):
    """return the conjunction of the input formulas"""
    return '{} \u2227 {}'.format(formula_a, formula_b)


def Disjunction(formula_a: str, formula_b: str):
    """return the disjunction of the input formulas"""
    return '{} \u2228 {}'.format(formula_a, formula_b)


def Implication(antecedent_formula: str, consequent_formula: str):
    """return the implication formula of the
    antecedent formula and consequent formula"""
    return '{} \u2192 {}'.format(
        antecedent_formula, consequent_formula)


def Equivalence(formula_a: str, formula_b: str):
    """return the logical equivalence formula of
    the input formulas"""
    return '{} \u2194 {}'.format(formula_a, formula_b)


def Nonequivalence(formula_a: str, formula_b: str):
    """return the logical non-equivalence formula of
    the input formulas"""
    return '{} \u2295 {}'.format(formula_a, formula_b)


def ExistentialQuantification(formula: str, variable_symbol: str):
    """return an existential quantification of the input formula
    and the input variable symbol"""
    assert variable_symbol in formula
    return '\u2203{}({})'.format(variable_symbol, formula)


def UniversalQuantification(formula: str, variable_symbol: str):
    """return an universal quantification of the input formula
    and the input variable symbol"""
    assert variable_symbol in formula
    return '\u2200{}({})'.format(variable_symbol, formula)


def End(formula: str):
    return formula
```

C Algorithm of FOL Grammar Tree to Python Code Sequence

The Python-style pseudo code of the FOL grammar tree to Python code sequence is shown in Algorithm 2.

Algorithm 2 Pseudo code for obtaining code sequence from a first-order logic grammar tree in Python-style.

```
class FOLGrammarTreeNode:
    def __init__(self, start, end, string, type):
        # start position in natural language statement
        self.start = start
        # end position in natural language statement
        self.end = end
        # string expression
        self.string = string
        # type, can be `variable`, `constant`, e.t.c.
        self.type = type
        # children node list
        self.children = []

# assign each subformula a unique index
expression2idx = {}
code_sequence = []

def construct_code_sequence(node: FOLGrammarTreeNode):
    # store the children node information
    children_attributes = []
    for child in node.children:
        children_attributes.append(
            construct_code_sequence(child))

    # Get the variable that carries the child node
        formula according to the different node.type,
        and then generate the code.
    code = get_code(node.type, children_attributes)
    code_sequence.append(code)
    idx = len(expression2idx) + 1
    expression2idx[node.string] = idx

    # return the information corresponding to current
        node
    return {
        'type': node.type,
        'string': node.string,
    }

construct_code_sequence(tree)
code_sequence.append('expression␣=␣End({})'.format(
    code_sequence[-1].split('=')[0].strip()))
```

D More Details of Experiments

D.1 Implement Details

FOL Parse. To parse the first-order logical formula into a tree structure, we develop the FOL parser using Pyleri[3]. Pyleri is a user-friendly parser

[3] https://github.com/cesbit/pyleri

FOL Grammar

$$
\begin{aligned}
&< variable > ::= \texttt{variable\_string}\\
&< constant > ::= \texttt{constant\_string}\\
&< function\_name > ::= \texttt{function\_name\_string}\\
&< predicate\_name > ::= \texttt{predicate\_name\_string}\\
&< term > ::= < variable >\\
&\qquad\qquad \mid < constant >\\
&\qquad\qquad \mid < function\_name > \text{ `(' } < term > \{\text{`,' } < term > \} \text{ `)'}\\
&< atomic\_formula > ::= \text{ `True'}\\
&\qquad\qquad\qquad\qquad \mid \text{`False'}\\
&\qquad\qquad\qquad\qquad \mid < term > = < term >\\
&\qquad\qquad\qquad\qquad \mid < term > \neq < term >\\
&\qquad\qquad\qquad\qquad \mid < predicate\_name > \text{ `(' } < term > \{\text{`,' } < term > \} \text{ `)'}\\
&< formula > ::= < atomic\_formula >\\
&\qquad\qquad\quad \mid \text{`}\neg\text{' } < formula >\\
&\qquad\qquad\quad \mid < formula > \text{ `}\wedge\text{' } < formula >\\
&\qquad\qquad\quad \mid < formula > \text{ `}\vee\text{' } < formula >\\
&\qquad\qquad\quad \mid < formula > \text{ `}\oplus\text{' } < formula >\\
&\qquad\qquad\quad \mid < formula > \text{ `}\rightarrow\text{' } < formula >\\
&\qquad\qquad\quad \mid < formula > \text{ `}\leftrightarrow\text{' } < formula >\\
&\qquad\qquad\quad \mid \text{`(' } < formula > \text{ `)'}\\
&\qquad\qquad\quad \mid \text{`}\forall\text{' } < variable > < formula >\\
&\qquad\qquad\quad \mid \text{`}\exists\text{' } < variable > < formula >
\end{aligned}
$$

Figure 8: The BNF grammar of first-order logic. An item with a `string` suffix represents a character (*e.g.*, letters and common punctuation) terminal symbol.

build tool and can readily export the tree structure for subsequent code generation.

Code-LLMs. For Code-LLMs, we aim to conduct experiments with Codex from OpenAI aligning with previous works (Wang et al., 2023; Li et al., 2023b). However, the Codex models have been deprecated from the OpenAI APIs. Consequently, we opt to use the `gpt-3.5-turbo-16k` model from OpenAI for our experiments. Additionally, to facilitate a comparison of the performance among various code-LLMs, we also conduct experiments on the open-source code completion model CodeGeeX2. CodeGeeX2 (Zheng et al., 2023a) is constructed on the ChatGLM2 (Du et al., 2022; Zeng et al., 2023) architecture and trained on a more extensive dataset of code. For `gpt-3.5-turbo-16k`, we use the official API[4] to obtain model results, whereas we employ the open-source model parameters for CodeGeeX2[5]. During the generation process, the `temperature` is set to 0.7, the `max_tokens` is set to 500, and other parameters are kept at default values.

Construction of Demonstration Examples. In our experiments, the construction of demonstration examples is mainly based on the public datasets FOLIO (Han et al., 2022) and MALLS (Yang et al., 2024b). FOLIO, curated by expert annotators, functions as a natural language reasoning dataset that presents new challenges in first-order logical reasoning. It offers a wide array of natural language variations, an extensive vocabulary, and diverse logic patterns, comprising 8k NL-FOL pairs. Con-

[4] https://openai.com/api

[5] https://huggingface.co/THUDM/codegeex2-6b

Prompt of CODE4LOGIC

```
'''
Please utilize the functions provided below to systematically generate the
first-order logic formula that corresponds to the natural language statement.
'''
<There are basis function definition>

natural_language_statement = 'If a convicted criminal is found guilty' +\
', then they are sentenced to a punishment.'
formula1 = Variable('x')
formula2 = Predicate('Convictedcriminal ', [formula1])
formula3 = Predicate('Foundguilty ', [formula1])
formula4 = Predicate('Sentencedtopunishment ', [formula1])
formula5 = Conjunction(formula2, formula3)
formula6 = Implication(formula5, formula4)
formula7 = UniversalQuantification(formula6, formula1)
formula = End(formula7)
<There are K demonstration examples>

natural_language_statement = <query natural language statement>
```

Figure 9: The detailed prompt of CODE4LOGIC.

Prompt of Few-Shot Baselines

Please convert the statement from natural language into first-order logical formula.

natural language statement: All heavy things are still.
first-order logic formula: $\forall$x (Heavy(x) $\rightarrow$ Still(x))
<There are K demonstration examples>

natural language statement: <query natural language statement>
first-order logic formula:

Figure 10: The detailed prompt of LLM-Based few-shot baselines.

versely, MALLS is an NL-FOL dataset generated by GPT-4 (OpenAI, 2023). Compared to FOLIO, MALLS introduces a broader range of contextually rich NL-FOL pairs, encompassing 34K instances. In practice, we merge all NL-FOL pairs from the training sets of FOLIO and MALLS, converting them into code sequence format to construct the supporting dataset $\mathcal{D}$. During in-context learning, we randomly sample a fixed number (3 or 5, based on the input constraints of the Code-LLMs) of samples from $\mathcal{D}$ for each basis function to serve as demonstration examples.

D.2 Full Prompt of CODE4LOGIC

We provide the detailed prompt of CODE4LOGIC in Figure 9.

D.3 NL-FOL Translation Baselines

D.3.1 LLM-Based Few-Shot Baselines

For all of the large language model based few-shot baseline models (*e.g.*, Claude-1, GPT4, text prompt based CodeGeeX, and GPT3.5), we use the following prompt template as shown in Figure 10.

When generating, the `temperature` is set to 0.7, the `max_tokens` is set to 200, and other parameters are kept at default values.

Prompt of Few-Shot Baselines For LogicNLI

Given the premise and hypothesis, please determine the relationship between the two, and choose from the following options:
A: contradiction. B: self_contradiction. C: neutral. D: entailment.

<There are K demonstration examples>

premise: <query premise>
hypothesis: <query hypothesis>
choice:

Prompt of Few-Shot Baselines For RuleTaker

Given context and question, please determine whether the question can be entailment by the context and output yes or no.

<There are K demonstration examples>

context: <query context>
question: <query question>
output:

Prompt of Few-Shot Baselines For VitaminC

Given evidence and claim, please determine whether the claim can be supported by evidence, choose from the following options:
A: SUPPORTS. B: NOT ENOUGH INFO. C: REFUTES.

<There are K demonstration examples>

evidence: <query evidence>
claim: <query claim>
choice:

Figure 11: Prompt of few-shot baselines for downstream datasets.

D.3.2 Supervised Training Baselines

We mainly compare CODE4LOGIC with two prominent baselines:

- **LOGICLLAMA** (Yang et al., 2024b). LOGICLLAMA is a translation model for translating natural language to first-order logic based on LLaMA (Touvron et al., 2023a), fine-tuned with LoRA (Hu et al., 2022). The authors begin by curating a high-quality and diverse dataset consisting of NL-FOL pairs at the sentence level obtained from GPT-4. Subsequently, they generate a perturbed variation of the logical formula in each pair to establish a controlled perturbation dataset. Additionally, they introduce the innovative SFT+RLHF framework, which trains LOGICLLAMA on the artificially perturbed NL-FOL pairs.
- **Symbol-LLM** (Xu et al., 2024). Symbol-LLM comprises a series of models designed for text-to-symbol tasks. The authors undertake an extensive collection of 34 text-to-symbol generation tasks, encompassing approximately 20 standard symbolic forms introduced with instruction tuning format. Then they employ a two-stage continual tuning framework to tune the LLaMA-2-Chat (Touvron et al., 2023b) model.

In the study, we utilized the public model parameters of LOGICLLAMA and Symbol-LLM for the experiment, maintaining consistency with the generation hyperparameters as reported in the original paper.

D.4 Downstream Tasks Baselines

D.4.1 LLM-Based Few-Shot Baselines

All of the large language model-based few-shot baseline models evaluated on the LogicNLI, RuleTaker, and VitaminC datasets use the prompt template illustrated in Figure 11.

Similarly, the `temperature` is set to 0.7, the `max_tokens` is set to 200, and other parameters are kept at default values.

D.4.2 Supervised Training Baselines

We include different supervised training based baseline models on each of the three downstream task datasets.

- **LogicNLI.** Pretrained sequence-to-sequence models like BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and XLNet (Yang et al., 2019) are employed for solving the logical reasoning task. Each sample's premise and hypothesis from the dataset are concatenated and fed into the model. Then, a classifier head is utilized to make predictions.
- **RuleTaker.** In addition to RoBERTa (Liu et al., 2019) as the baseline model similar to LogicNLI, we introduce Neural Unifer (Picco et al., 2021), a novel architecture that emulates unification and fact-checking algorithms to enhance the generalization capability for responding to complex queries.
- **VitaminC.** We compare CODE4LOGIC with three salient supervised training based baselines: FactCC (Kryscinski et al., 2020), BLANC (Vasilyev et al., 2020), and BARTSCORE (Yuan et al., 2021). FactCC is a novel weakly supervised BERT-based model designed to verify factual consistency. The model is trained using synthesized data created from source documents through a set of rule-based transformations inspired by error analysis of outputs from state-of-the-art summarization models. BLANC is a BERT-based (Devlin et al., 2019) method for automatically assessing the quality of document summaries, which can also be utilized for evaluating factual consistency. BARTSCORE assesses factual consistency by framing it as a text generation task and resolves the modeling challenge using the Bart (Lewis et al., 2020) model.

For baselines with open-source code, we make direct use of the hyperparameters reported in the paper. For other baselines, we select the optimal hyperparameters on the validation set and report the results.

D.5 Dataset Statistics

The statistics of FOLIO and MALLS are summarized in Table 6 and the statistics of three downstream task datasets are shown in Table 7.

Dataset	#Train	#Valid	#Test
FOLIO	1,001	203	-
MALLS	27,284	-	1,000

Table 6: Statistics of FOLIO and MALLS.

Dataset	#Train	#Valid	#Test
LogicNLI	16,000	2,000	2,000
RuleTaker	480,152	75,872	151,911
VitaminC	370,653	63,054	55,197

Table 7: Statistics of LogicNLI, RuleTaker, and VitaminC.

D.6 Hardcore Configurations

We conducted all experiments in the following hardware environment:

- Operating System: Ubuntu 22.04.3 LTS.
- CPU: Intel Xeon Gold 6148 CPU @ 2.40GHz with 384GB DDR4 of Memory.
- GPU: NVIDIA Tesla A100 SMX4 with 80GB of Memory.
- Software: CUDA 11.8, Python 3.9.14, PyTorch (Paszke et al., 2019) 2.3.0.

E Additional Experimental Results

E.1 Performance w.r.t. Different LLMs

Table 8 illustrates the experimental results of CODE4LOGIC using different LLms. We can observe that even with non-code-tuned LLMs, Code4Logic can achieve prominent results.

Model	FOLIO	MALLS
CodeGeeX (Zheng et al., 2023b)	84.77	85.81
GPT3.5-turbo-16k	92.67	90.92
LLaMA3-8b (Dubey et al., 2024)	86.31	86.73
CodeLLaMA-7b (Rozière et al., 2023)	88.62	87.97

Table 8: Results on FOLIO and MALLS using different LLMs.

E.2 Case Study of CODE4LOGIC

The cases are shown in Figure 12. We randomly sample cases from LogicNLI, RuleTaker, and VitaminC.

F More Discussion about Studies on Semantic Parsing

Semantic parsing (Zettlemoyer and Collins, 2005; Liang, 2013; Andreas et al., 2013; Agrawal et al., 2019; Shin et al., 2021) involves transforming natural language into formal languages like lambda expressions, SQL, and graphs, which computers can understand. These formal languages are commonly used for creating queries and commands. Although these formalisms are quite direct, first-order logic presents a wider range of potential uses. Methodologically, our study investigates the viability of utilizing large language models in logical transformation, moving away from the traditional approach of semantic representation combined with parsing models, without the need for supplementary model training.

```
natural_language_statement = 'If all people are not dramatic, ' +\
'then Clyde is impatient and Clyde is not eager.'
formula1 = Variable('x')
formula2 = Predicate('Dramatic', [formula1])
formula3 = Negation(formula2)
formula4 = Constant('clyde')
formula5 = Predicate('Impatient, [formula4])
formula6 = Predicate('Eager', [formula4])
formula7 = Negation(formula6)
formula8 = Conjunction(formula5, formula7)
formula9 = Implication(formula3, formula8)
formula10 = UniversalQuantification(formula9, formula1)
formula = End(formula10)
```

$\forall x(\neg \mathtt{Dramatic}(x) \rightarrow \mathtt{Impatient}(\mathtt{clyde}) \wedge \neg \mathtt{Eager}(\mathtt{clyde}))$

```
natural_language_statement = 'Someone is eager if and only if he is not super.'
formula1 = Variable('x')
formula2 = Predicate('Eager', [formula1])
formula3 = Predicate('Super', [formula1])
formula4 = Negation(formula3)
formula5 = Equivalence(formula2, formula4)
formula6 = UniversalQuantification(formula5, formula1)
formula = End(formula6)
```

$\forall x(\mathtt{Eager}(x) \leftrightarrow \neg \mathtt{Super}(x))$

```
natural_language_statement = 'If someone is round then they are green.'
formula1 = Variable('x')
formula2 = Predicate('Round', [formula1])
formula3 = Predicate('Green', [formula1])
formula4 = Implication(formula2, formula3)
formula5 = UniversalQuantification(formula4, formula1)
formula = End(formula5)
```

$\forall x(\mathtt{Round}(x) \rightarrow \mathtt{Green}(x))$

```
natural_language_statement = 'Sky Sports Mexico has the rights to a ' +\
'couple of live matches.'
formula1 = Constant('skysportsmexico')
formula2 = Constant('livematches')
formula3 = Predicate('Hasrights', [formula1, formula2])
formula = End(formula3)
```

$\mathtt{Hasrights}(\mathtt{skysportsmexico}, \mathtt{livematches})$

```
natural_language_statement = 'Greg Kot was commented favorably on ' +\
'Ride the Lightning.'
formula1 = Constant('gregkot')
formula2 = Predicate('Commentfavorably', [formula1])
formula3 = Constant('ridethelightning')
formula4 = Predicate('On, [formula1, formula3])
formula5 = Conjunction(formula2, formula4)
formula = End(formula5)
```

$\texttt{Commentfavorably}(\texttt{gregkot}) \wedge \texttt{On}(\texttt{gregkot}, \texttt{ridethelightning})$

```
natural_language_statement = 'If Carrick is not noisy, then Arnold is not ' +\
'huge and Nick is intelligent.'
formula1 = Constant('carrick')
formula2 = Predicate('Noisy', [formula1])
formula3 = Negation(formula2)
formula4 = Constant('arnold')
formula5 = Predicate('Huge', [formula4])
formula6 = Negation(formula5)
formula7 = Implication(formula3, formula6)
formula = End(formula7)
```

$\neg\texttt{Noisy}(\texttt{carrick}) \rightarrow \neg\texttt{Huge}(\texttt{arnold})$

```
natural_language_statement = 'If there is someone who is not crystal, then ' +\
'Roger is not clean and Adley is not glamorous.'
formula1 = Variable('x')
formula2 = Predicate('Ctysral', [formula1])
formula3 = Negation(formula2)
formula4 = Constant('roger')
formula5 = Predicate('Clean', [formula4])
formula6 = Negation(formula5)
formula7 = Constant('adley')
formula8 = Predicate('Glamorous', [formula7])
formula9 = Negation(fromula7)
formula10 = Conjunction(formula6, formula9)
formula11 = Implication(formula3, formula10)
formula = End(formula11)
```

$\forall x(\neg\texttt{Crystal}(x) \rightarrow \neg\texttt{Clean}(\texttt{roger}) \wedge \neg\texttt{Glamorous}(\texttt{adley}))$

Figure 12: Cases of CODE4LOGIC.