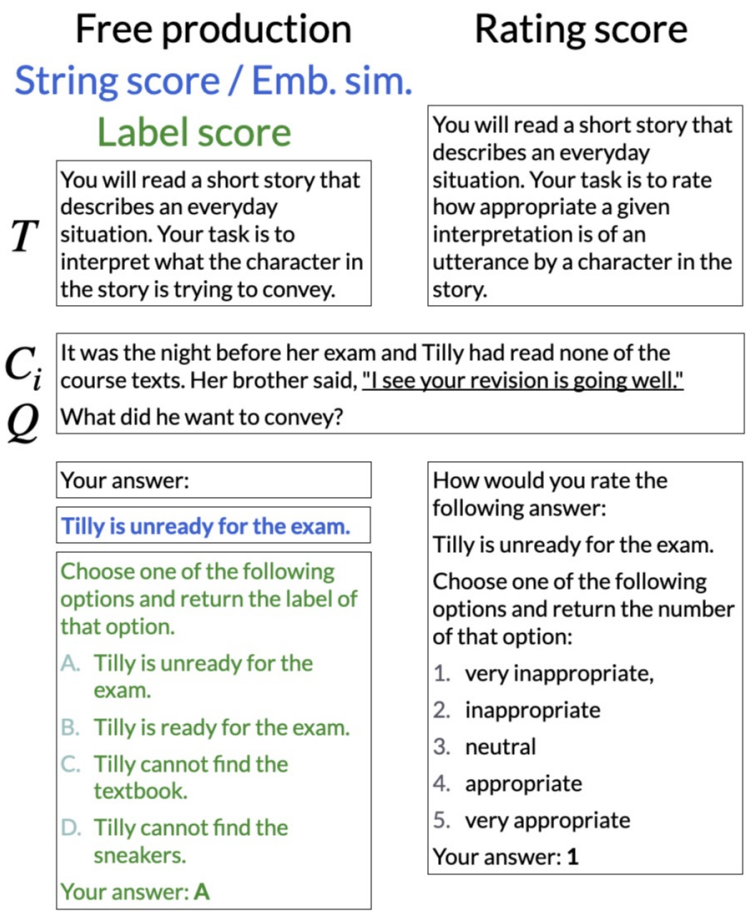
Assessing LLM Performance

Scott

Methods



- Free generation
- String score
- Label score
- Embedding Similarity

Guided Generation / Structured Output

- JSONFORMER

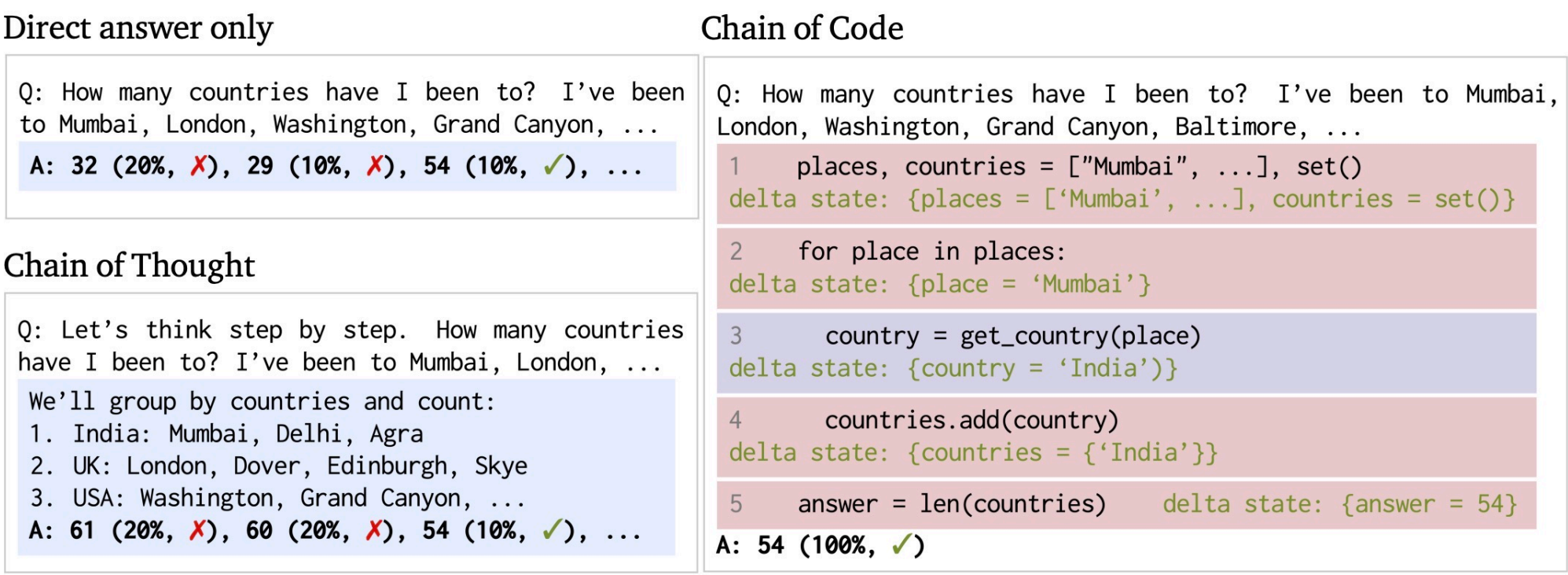
A Bulletproof Way to Generate Structured JSON from Language Models.

https://github.com/1rgs/jsonformer/

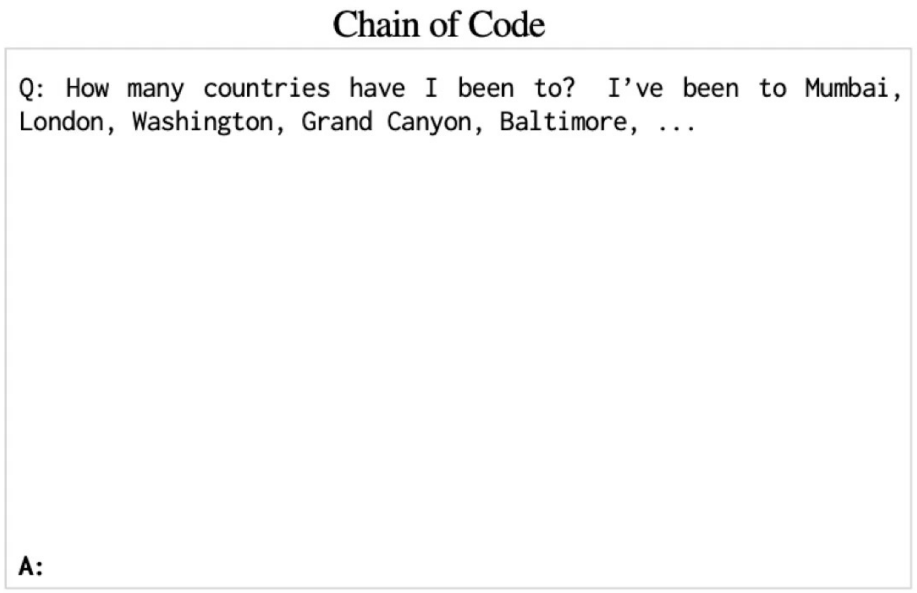
- GUIDANCE

https://github.com/guidance-ai/guidance

Chain of Code: Reasoning with a Language Model-Augmented Code Emulator



Chain of Code: Reasoning with a Language Model-Augmented Code Emulator



Blue highlight indicates LM generation.
Red highlight indicates LM generated code being executed by an interpreter.
Purple highlight indicates an LMulator simulating the code via a program state in green.

(a) Chain of Thought [42]

```
Q: Roger has 5 balls. He buys 2 more packs, each with 3. How many balls does he have now?

Roger starts with 5 balls.
2 packs of 3 balls is 6.
5 + 6 = 11.

A: 11
```

(b) Program of Thoughts [5]

```
Q: Roger has 5 balls. He buys 2 more packs, each with 3. How many balls does he have now?

num_balls = 5
num_balls += 2 * 3
answer = num_balls

A: 11
```

(c) ScratchPad [26]

```
Q: Roger has 5 balls. He buys 2 more packs, each with 3. How many balls does he have now?

num_balls = 5    state: {num_balls = 5}
num_balls += 2 * 3    state: {num_balls = 11}
answer = num_balls    state: {answer = 11}

A: 11
```

(d) Chain of Code Generation

```
Q: I have an orange, a violin, two peaches, an apple, a pepper, and three plums. How many fruits do I have?

1  objects = {"orange": 1, "violin": 1, "peaches": 2, "apple": 1, "pepper": 1, "plum": 3}
2  num_fruits = 0
3  for object in objects:
4    object_is_fruit = is_fruit(object)
5    if object_is_fruit:
6      num_fruits += objects[object]
7  answer = num_fruits
```

(e) Chain of Code Execution

```
Q: I have an orange, a violin, two peaches, an apple, a pepper, and three plums. How many fruits do I have?

1  objects = {"orange": 1, "violin": 1, "peaches": 2, "apple": 1, "pepper": 1, "plum": 3}
delta state: {objects = {'orange': 1, 'violin': 1, ...}}
2  num_fruits = 0
delta state: {num_fruits = 0}
3  for object in objects:
delta state: {object = 'orange'} # updated for each loop
4    object_is_fruit = is_fruit(object)
delta state: {object_is_fruit = True}
5    if object_is_fruit:
delta state: {}
6      num_fruits += objects[object]
delta state: {num_fruits = 1}
7  answer = num_fruits
delta state: {answer = 7}

A: 7
```

Figure 2 | **Previous reasoning methods:** To solve advanced problems, (2a) Chain of Thought prompting breaks the problem down into intermediate steps, (2b) Program of Thoughts prompting writes and executes code, and (2c) ScratchPad prompting simulates running already written code by tracking intermediate steps through a program state. **Our reasoning method:** Chain of Code first (2d) generates code or psuedocode to solve the question and then (2e) executes the code with a code interpreter if possible, and with an LMulator (language model emulating code) otherwise. Blue highlight indicates LM generation, red highlight indicates LM generated code being executed, and purple highlight indicates LMulator simulating the code via a program state in green.

Tasks

- BBH

Semantic reasoning

Numerical reasoning

- GSM8K

(c) Logical Deduction

```
Q: The following paragraphs each describe a set of three objects arranged in a fixed order. The statements are logically consistent within each paragraph. On a shelf, there are three books: a green book, a red book, and a blue book. The red book is the rightmost. The blue book is to the right of the green book.
Options:
(A) The green book is the leftmost
(B) The red book is the leftmost
(C) The blue book is the leftmost
```

```
options = {"green": "(A)", "red": "(B)", "blue": "(C)"}
delta state: {options = {'green': ..., ..., 'blue': ...}}
```

```
order_info = "left to right"
delta state: {order_info = 'left to right'}
```

```
full_order = [None, None, None]
delta state: {full_order = [None, None, None]}
```

```
partial_order = []
delta state: {partial_order = []}
```

```
full_order[-1] = "red"
delta state: {full_order = [None, None, 'red']}
```

```
partial_order.append(("green", "blue"))
delta state: {partial_order = [('green', 'blue')]}
```

```
full_order = generate_full_order(full_order, partial_order, ret_type=list)
delta state: {full_order = ['green', 'blue', 'red']}
```

```
query = "leftmost"
delta state: {query = 'leftmost'}
```

```
result = query_result(order_info, full_order, query, ret_type=str)
delta state: {result = 'green'}
```

```
answer = options[result] if result in options else None
delta state: {answer = '(A)'}
```

- BBH

Semantic reasoning

Numerical reasoning

- GSM8K

- Base Model

Text-ada-001

Text-baggage-001

Text-curie-001

Text-davinci-003

- Code Model

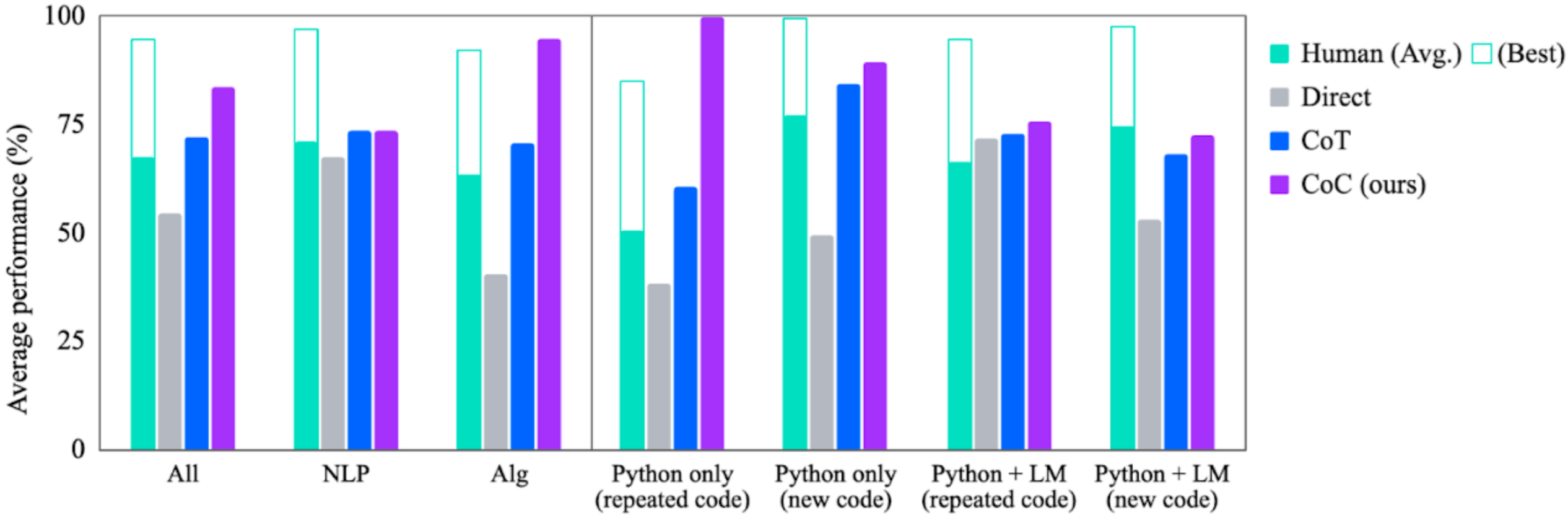
PaLM-2

- Instruction-tuned model

Gpt-3.5-turbo

Gpt-4

text-davinci-003	gpt-3.5-turbo				gpt-4			
CoC (Interweave)	Direct	CoT	CoC (Python)	CoC (LM)	Direct	CoT	CoC (Python)	CoC (LM)
84	51 (-33)	56 (-28)	56 (-28)	45 (-39)	70 (-14)	78 (-6)	82 (-2)	75 (-9)



LINC: A neuro-symbolic approach for logical reasoning by combining language models with first-order logic provers

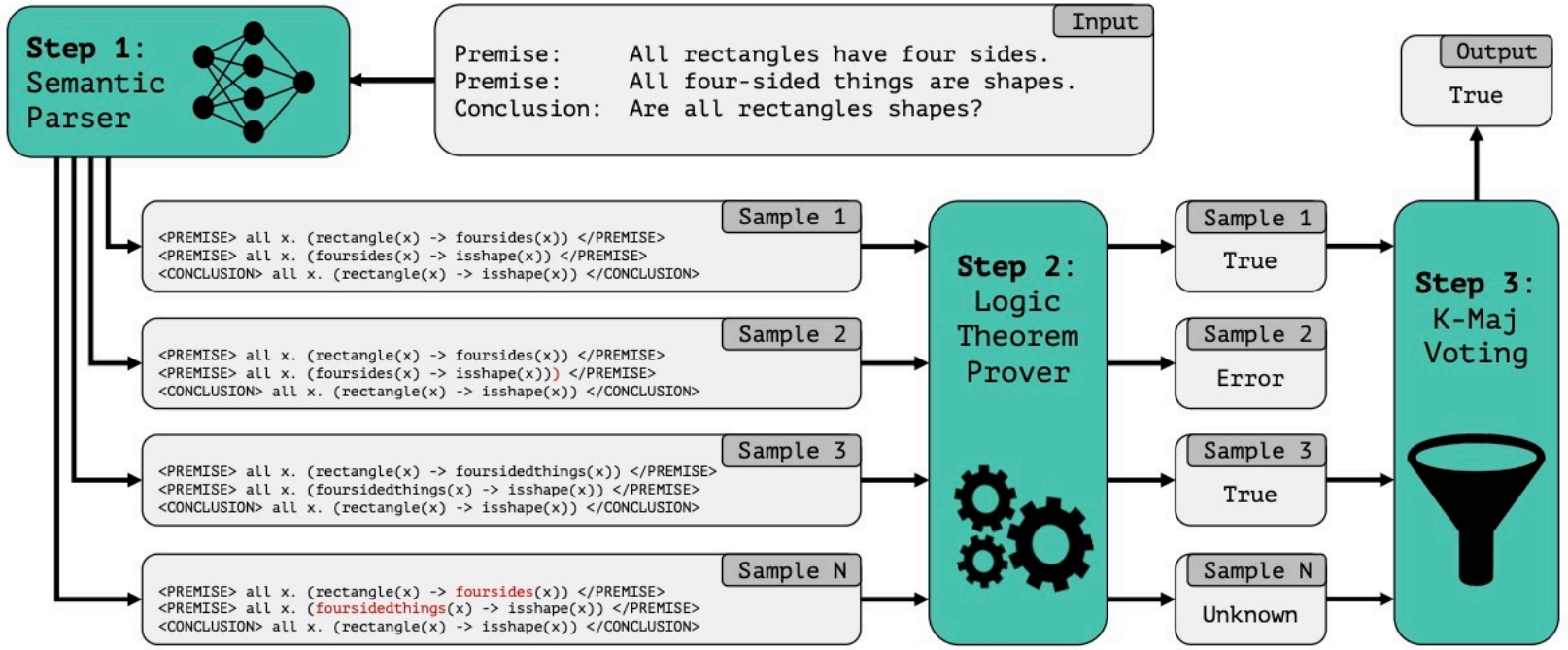


Figure 1: This figure showcases the essence of our approach. Starting from a problem in natural language, in **Step 1**, the LLM semantic parser samples logic formulas expressing estimates of the semantics. It is possible that some of these might contain errors, e.g., the second example shows a syntax error involving an extra parenthesis, whereas the fourth example highlights a semantic error caused by mismatched predicates. In **Step 2**, these are then each offloaded to an automated theorem prover, filtering out syntax errors, and producing labels for the remaining samples. In **Step 3**, the remaining candidate outputs are passed through a majority-vote sieve to arrive at the best estimate for a single output label.

- FOLIO
- ProofWriter

- StarCoder+
- GPT-3.5
- GPT-4

Prove9

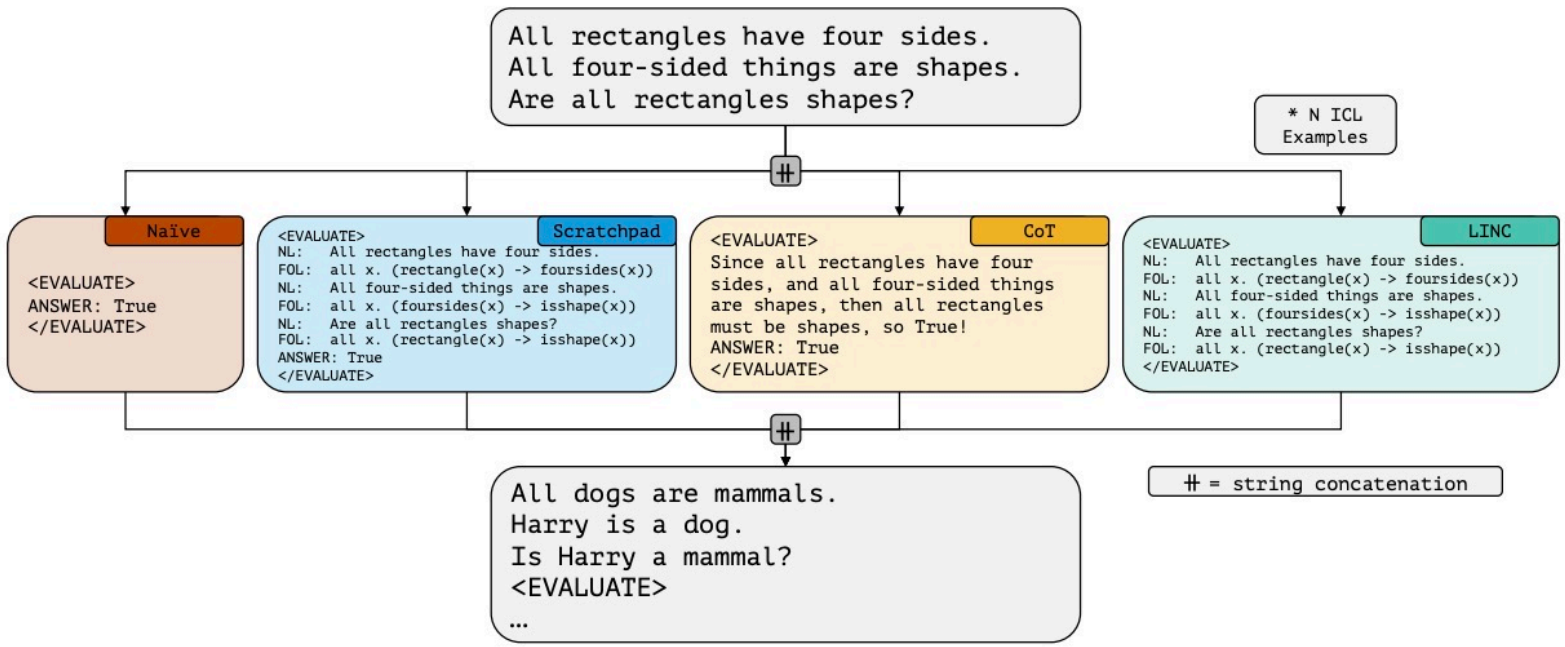
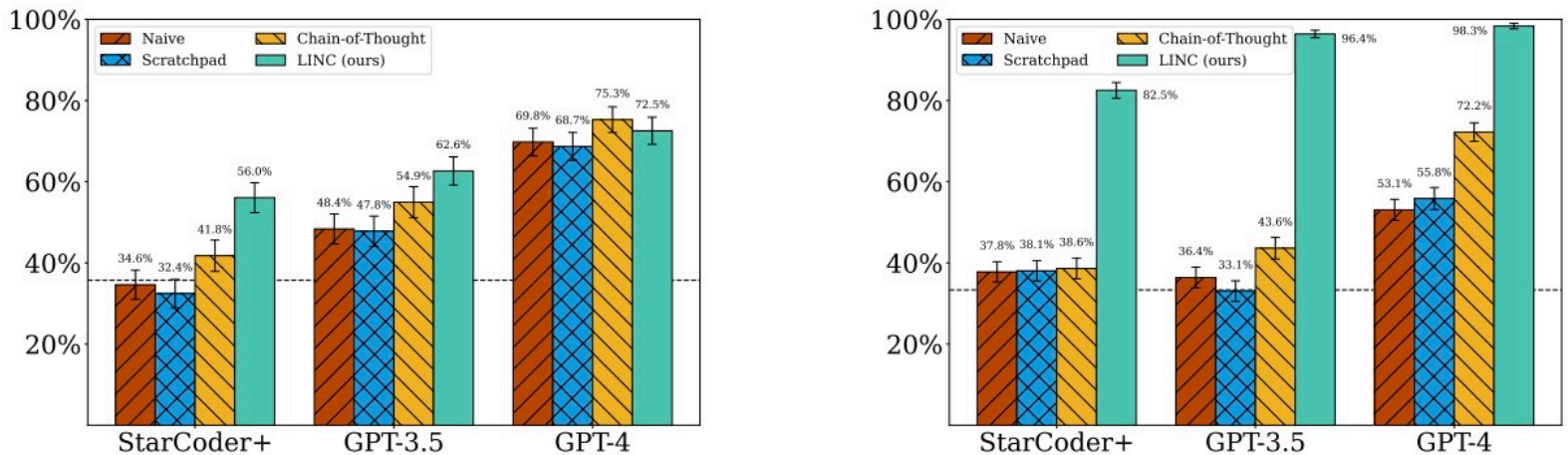


Figure 2: This figure outlines the string concatenation workflow for each of our conditions. We start with the original problem, provide ICL examples through an intermediate markup language, and finally append the problem to evaluate. At this stage, we allow the model to autoregressively sample until producing a stop token.



(a) FOLIO.

(b) ProofWriter.

Figure 3: Results of each model on the FOLIO and ProofWriter datasets. Accuracies are for bootstrapped 10-way majority vote for all models. Error bars are ± 1 bootstrapped standard deviation. Dotted, black line is the accuracy obtained by always guessing the most common label in the dataset.

Logic-Driven Context Extension and Data Augmentation for Logical Reasoning of Text

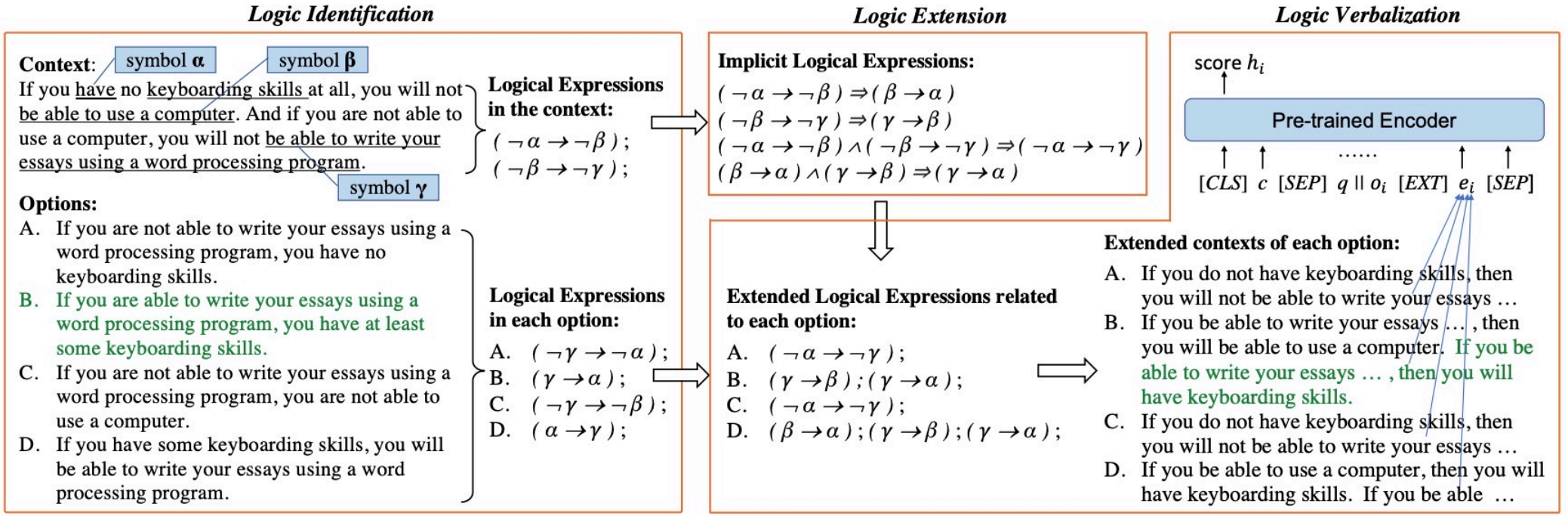


Figure 2: The overall architecture of our proposed logic-driven context extension framework. c, q, o_i and e_i are the context, question, i-th option and the extended context for i-th option, respectively. The texts in green mean that the option B is matched against its extended context which has the highest score.

- ReClor

- RoBERTa
- ALBERT