# Recurrent Inference in Text Editing

**Ning Shi**
**ning.shi@gatech.edu**

Ziheng Zeng
zzeng13@illinois.edu

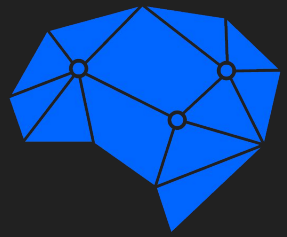
Haotian Zhang
haotian.zhang@learnable.ai

Yichen Gong
yichen01.gong@horizon.ai

Georgia Tech

UIUC

Learnable.ai

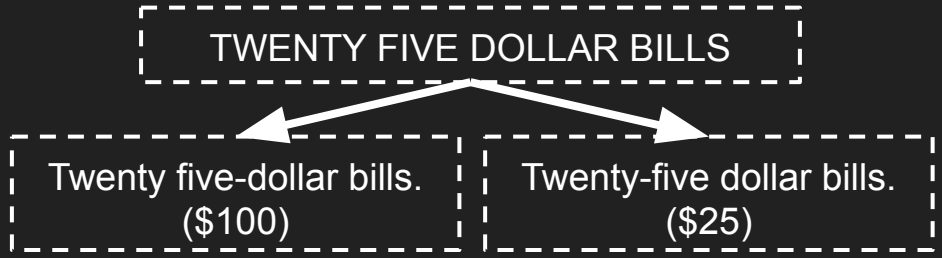


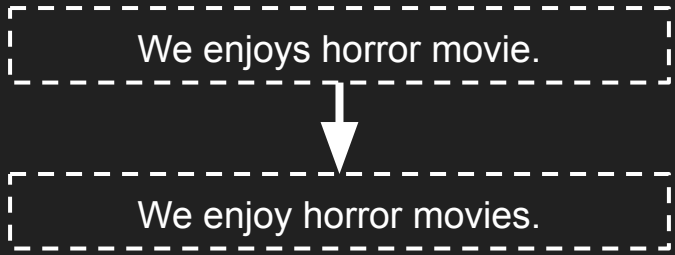
Horizon Robotics

# Text Editing

There is a high overlap between source text and target text.

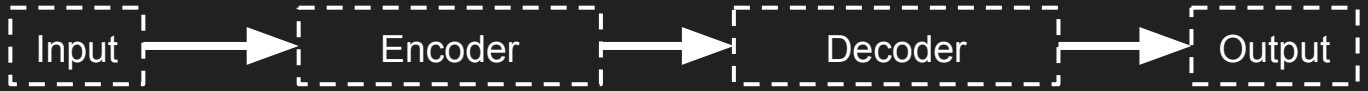
- Types (length relationship)
  - Short-to-long (become longer)
  - Long-to-short (become shorter)
  - Mixed (both cases)
- Tasks
  - Abstractive summarization
  - Text post-processing
  - Punctuation restoration
  - Grammar correction
  - Etc.
- Operations (Actions)
  - Keep
  - Delete
  - Insert
  - Substitute
  - Etc.



# End2end

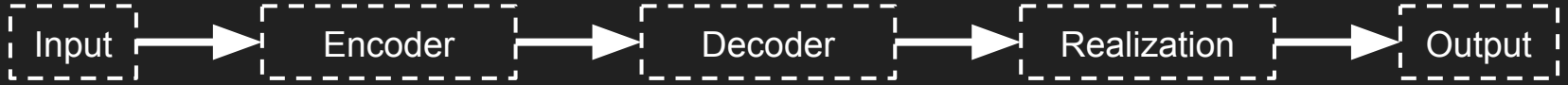
End-to-end is one of the early methods to perform text editing by casting the job as sequence-to-sequence (seq2seq) text generation (Sutskever et al., 2014).




# Tagging

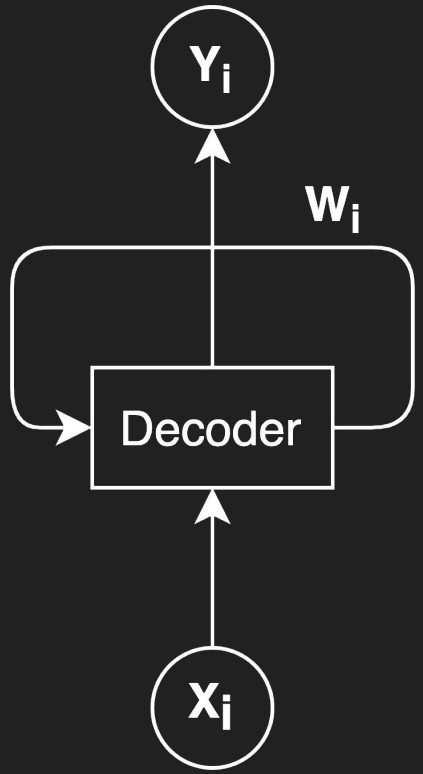
Tagging solves text editing in two steps.

1. Produces tag sequences (seq2seq, Neural Programmer-Interpreter by Reed and de Freitas, 2016)
2. Edits input texts according to the tag sequences (“realization” named by Malmi et al., 2019)



# Limitations

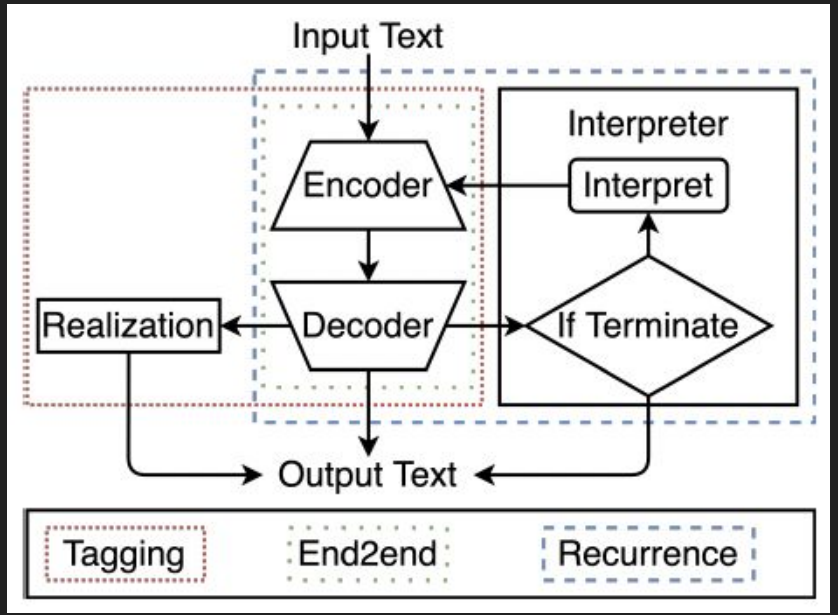
- End2end
  - Dependency on large amounts of data
  - Unexplainable processes
  - Unnecessary for text editing with high overlap
- Tagging
  - Large vocabulary to combine tags and words
  - Too many decoding steps to assign tags
- General
  - Static encoder hidden states (not up to date)
  - Massive pressure on the decoder
  - Various target sequence length (<EOS>)



# Recurrent Inference (Recurrence)

Recurrence divides a text editing task into multiple independent sub-tasks and completes them recurrently.

- Programmer
  - Determines editing actions
  - Encoder-decoder structure
- Editing Actions
  - The type of editing operation
  - The position the editing occurs
  - A text symbol
  - E.g., [insert, pos_1, A]
- Interpreter
  - A parameter-free function
  - Checks termination
  - Executes editing actions



# Recurrent Inference (Recurrence)

Recurrence divides a text editing task into multiple independent sub-tasks and completes them recurrently.

- Programmer
  - Determines editing actions
  - Encoder-decoder structure
- Editing Actions
  - The type of editing operation
  - The position the editing occurs
  - A text symbol
  - E.g., [insert, pos_1, A]
- Interpreter
  - A parameter-free function
  - Checks termination
  - Executes editing actions

**Algorithm 1:** Recurrence

**Result:** $\mathbf{y}^{(\text{complete})}$

$\mathbf{x}_{\text{Input}} = \mathbf{x}$;

Terminate = False;

$t = 0$;

**while** *Terminate is not True* **do**

  $t = t + 1$;

  $\mathbf{a}^{(t)} = \texttt{Programmer}(\mathbf{x}_{\text{Input}})$;

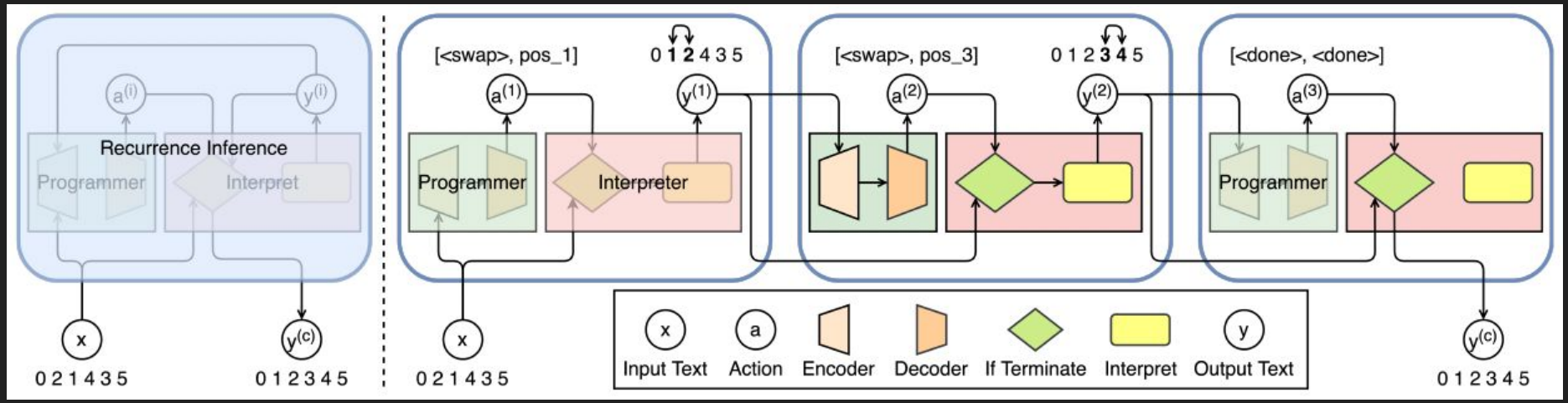
  $\mathbf{y}^{(t)}$, Terminate = $\texttt{Interpreter}(\mathbf{x}_{\text{Input}}, \mathbf{a}^{(t)})$ ;

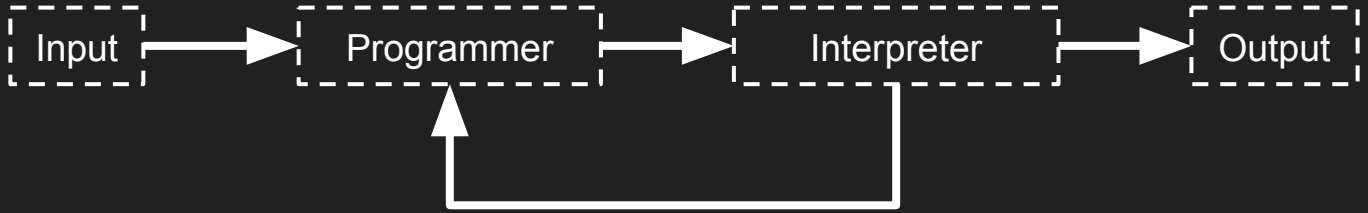
  $\mathbf{x}_{\text{Input}} = \mathbf{y}^{(t)}$;

**end**

$\mathbf{y}^{(\text{complete})} = \mathbf{y}^{(t)}$;

# Recurrent Inference (Recurrence) - Sorting



# Arithmetic Equation Problem

We design three easily reproducible, proof-of-concept text editing tasks

1. Arithmetic Operator Restoration (AOR)
2. Arithmetic Equation Simplification (AES)
3. Arithmetic Equation Correction (AEC)

We describe an arithmetic equation dataset from three aspects:

1. N - the number of unique integers (control the vocabulary size)
2. L - the number of integers in an equation (control the sequence length)
3. D - the number of unique equations (control the data size)

# Arithmetic Operators Restoration

- Goal
  To convert a sequence of integer numbers into a valid arithmetic equation.
- One-to-many
  Each integer sequence potentially corresponds to different valid arithmetic equations.
- Short-to-long
  The sequence length always go from short to long.

| | **AOR** ($N = 10$, $L = 6$) |
|---|---|
| Input | $6 \;\; 10 \;\; 9 \;\; 5 \;\; 2 \;\; 3$ |
| Output | $-6 \;/\; 10 + 9 \;/\; 5 * 2 == 3$ |
| Input | $2 \;\; 2 \;\; 4 \;\; 8 \;\; 2 \;\; 4$ |
| Output | $2 * 2 - 4 + 8 \;/\; 2 == 4$ |
| Input | $2 \;\; 2 \;\; 4 \;\; 8 \;\; 2 \;\; 4$ |
| Output | $-2 + 2 \;/\; 4 * 8 + 2 == 4$ |

# Arithmetic Equation Simplification

- Goal
  To simplify equations by calculating the parts in parentheses
- Many-to-one
  Variants share the same simplified form.
- Long-to-short
  The sequence length always go from long to short.

| | **AES** ($N = 10, L = 4$) |
|---|---|
| Input | $-\ 3 + 10\ /\ 2 == 2$ |
| Output | $-\ 3 + 10\ /\ 2 == 2$ |
| Input | $(-\ 2 + 4\ )\ /\ 7 * 7 == 2$ |
| Output | $2\ /\ 7 * 7 == 2$ |
| Input | $2\ /\ 7 * (\ 11 - 4\ ) == (\ 4 - 2\ )$ |
| Output | $2\ /\ 7 * 7 == 2$ |

# Arithmetic Equation Correction

- Goal
  To detect and correct possible mistakes for valid arithmetic equations.
- Many-to-many
  - We can generate many wrong equations based on one correct one.
  - A wrong equation can be modified into multiple correct equations
- Mixed
  The sequence length can be either shorter or longer.

| | **AEC** ($N = 10$, $L = 5$) |
|---|---|
| Input | $4 - 3 \ / \ 6 * 4 == 2$ |
| Output | $4 - 3 \ / \ 6 * 4 == 2$ |
| Input | $6 \ \ 7 * + \ / \ 7 + \ \ / \ 7 == 2$ |
| Output | $- \ 7 * 5 \ / \ 7 + 7 == 2$ |
| Input | $- \ 6 \ \ 5 + 11 - 2$ |
| Output | $- \ 6 + 5 + 11 - 8 == 2$ |

# Example Target Sequences

| | **AOR** ($N = 10, L = 5$) | **AES** ($N = 100, L = 5$) | **AEC** ($N = 10, L = 5$) |
|---|---|---|---|
| Source | 8 2 8 4 2 | $-\ 33 + 25 + 75 - 60 == (\ 30 - 23\ )$ | $7 * 8\ /\ 4\ 8\ 2 - == 6$ |
| $\text{Target}_{\text{End2end}}$ | $-\ 8 * 2\ /\ 8 + 4 == 2$ | $-\ 33 + 25 + 75 - 60 == 7$ | $7 * 8\ /\ 4 - 8 == 6$ |
| $\text{Target}_{\text{Tagging}}$ | <insert_−><keep><insert_*><keep> | <keep><keep><keep><keep><keep> | <keep><keep><keep><keep> |
| | <insert_/><keep><insert_+><keep> | <keep><keep><keep><keep><sub_7> | <keep><delete><sub_−><sub_8> |
| | <insert_==><keep> | <delete><delete><delete><delete> | <keep><keep> |
| $\text{Target}_{\text{Recurrence}}$ | <pos_0> − | <pos_9><pos_13> 7 | <delete><pos_5><pos_5> |

End2end

- The target text sequence.

Tagging

- The target tag sequence to realize the target text.

Recurrence Programmer

- The immediate action for the current text state.

# Offline Training & Online Training

Offline Training for End2end and Tagging
We name the training mode used by the conventional methods offline training.

- Unedited text sequences are paired with target text sequences directly. (End2end)
- Unedited text sequences are paired with target tag sequences before realizing the target text sequences. (Tagging)

Online Training for Recurrence

1. To train the programmer, we compute all intermediate actions that are required to edit source to target.
2. We uniformly sample one source-action pair from this intermediate list as the training data instance.

Fair Competition

- Online End2end & Online Tagging
  - Intermediate training instances are exposed to End2end and Tagging.
- Offline Recurrence
  - Only the immediate editing actions are fed to Recurrence.

# Experiments - Settings

Data
- 70% for training, 15% for validation, 15% for testing

Model
- After testing Transformer and a range of modern RNNs, we focus on the overall best-performed architecture, bidirectional LSTM (Schuster and Paliwal, 1997; Hochreiter and Schmidhuber, 1997) with an attention mechanism (Luong et al., 2015).
- Adam optimizer (Kingma and Ba, 2015)

Evaluation
- Token Accuracy - correct predictions at the token-level divided by the target sequence length
- Sequence Accuracy - correct predictions at the sequence-level divided by the test size
- Equation Accuracy - the number of true predicted equations divided by the test size

Training
- Single GeForce RTX Titan
- Early stopping (Prechelt, 1998) with a patience of 512 epochs.

# Experiments - Results

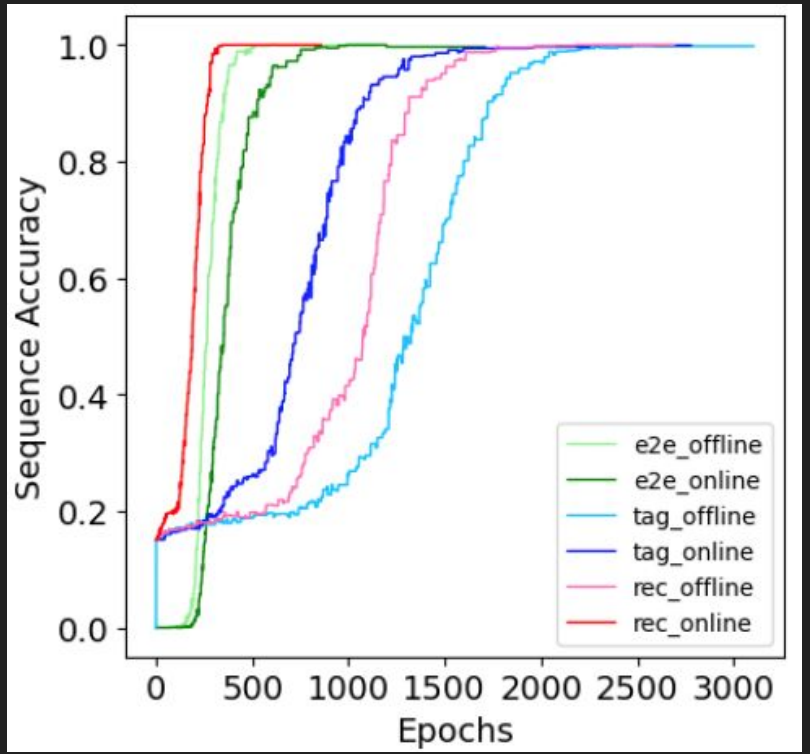
| | | **AOR** ($N = 10$, $L = 5$, $D = 10$K) | | **AES** ($N = 100$, $L = 5$, $D = 10$K) | | | **AEC** ($N = 10$, $L = 5$, $D = 10$K) | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Method** | **Training** | **#Epoch** | **Equ Acc.%** | **#Epoch** | **Token Acc.%** | **Seq Acc.%** | **#Epoch** | **Token Acc.%** | **Seq Acc.%** | **Equ Acc.%** |
| End2end | Offline | 3352 | 26.47 | 5063 | 75.49 | 3.27 | 72144 | 87.78 | 54.67 | 55.13 |
| | Online | 2640 | 29.33 | 7795 | 84.60 | 25.20 | 112482 | **88.08*** | 57.27 | 57.73 |
| Tagging | Offline | 1149 | 50.53 | 5223 | 90.10 | 43.80 | 135729 | 82.29 | 44.20 | 44.40 |
| | Online | 2245 | 51.40 | 4520 | 87.00 | 36.67 | 112968 | 84.46 | 46.93 | 47.33 |
| Recurrence | Offline | 1281 | 31.13 | 7603 | 94.92 | 62.07 | 203067 | 81.85 | 55.87 | 56.20 |
| | Online | 1898 | **58.53*** | 7088 | **98.63*** | **87.73*** | 152982 | 83.64 | **57.47*** | **58.27*** |

- Generally, Recurrence can achieve superior or competitive performances.
- Online training is critical for Recurrence to get better outcome.

# Experiments - Results

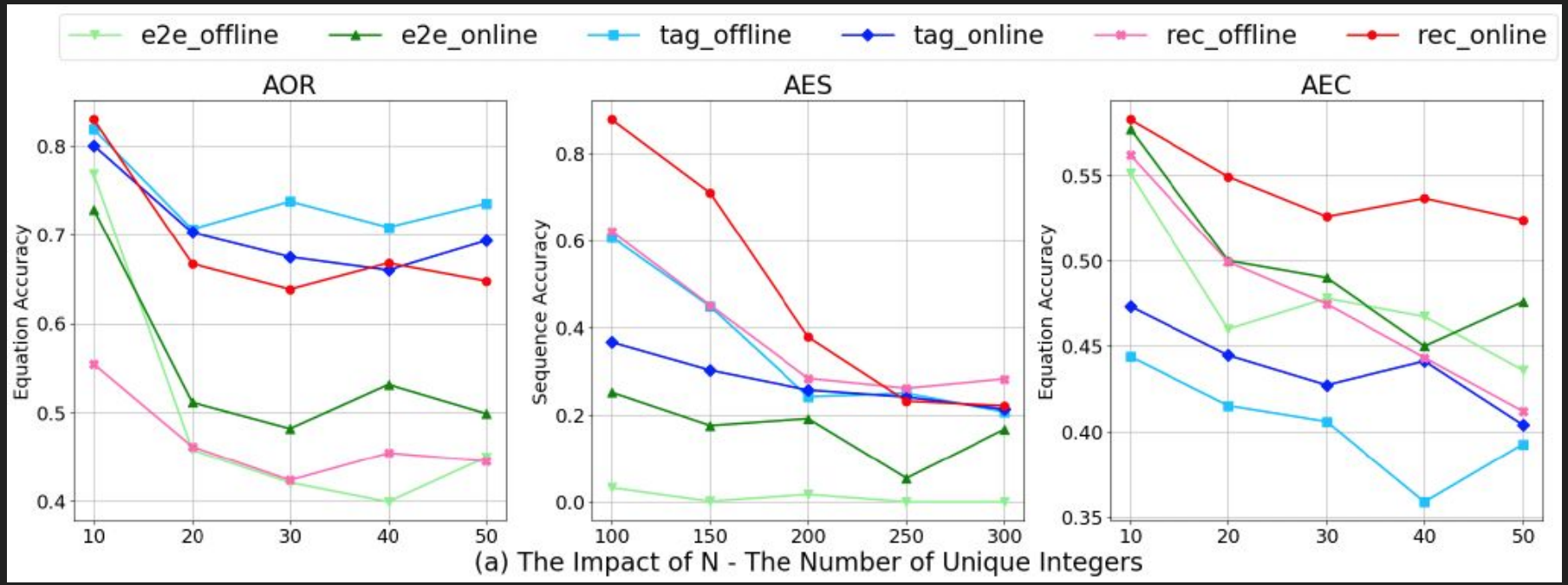
Figure shows testing sequence accuracy per epoch in AES with N = 10, L = 5, and D = 10K.

All methods can achieve near-perfect results, but online Recurrence converges the fastest.



# Analysis

We also explored the limits of Recurrence by running experiments with varying values of N, L and D, so as to determine in what scenario Recurrence performs well. Please see Figure 4 in our paper.



# Analysis - Some Observations

- Online Training
  - For End2end and Tagging, the online training acts like a data augmentation technique.
  - Offline training also allows Recurrence to gain some editing ability, at times better than End2end and Tagging.
  - For some tasks (AES), showing the immediate editing actions is enough for the model to generalize proper editing actions.
  - Letting the programmer produce one single editing step reduces the learning difficulty.

- Ordering
  - Programmer cannot converge if the data guide it to edit a sequence in a random order (a mixture of both left-to-right and right-to-left).
  - Ordering matters for not only text generation (Ford et al., 2018) but also Recurrence in test editing.
  - Random ordering may assign various actions to the same text state, and thus causes confusion.

# Conclusions

- Overall
  - Many text editing tasks can be solved by multiple inference steps recurrently
  - A novel recurrent inference method, Recurrence
  - Three easily reproducible, proof-of-concept text editing tasks, AOR, AES and AEC
  - Superior or competitive performances over the other two conventional methods
- Future Work
  - Apply Recurrence to open-domain natural language data
  - Investigate on how to relax its need for intermediate editing steps
  - Use pointer attention (Vinyals et al., 2015) to replace the position component in actions

# Q&A

Code: https://github.com/ShiningLab/Recurrent-Text-Editing.