

Investigating Neural Networks for Formal Language Recognition



Naisha Bhandari, Caroline Griem, Dr. Junyi Tu, Dr. Enyue Lu
Georgia Institute of Technology, Vassar College, Salisbury University



Background

A formal language is a set of strings following a mathematical rule. Models are tasked with accepting or rejecting a given string as part of a formal language. The FLaRe benchmark (Butoi et al., 2025) contains 18 formal language recognition tasks organized into structural families. These include marker-based transformation tasks (e.g., verify that w' is a specific transform of w in a string of the form $w\#w'$), arithmetic verification tasks (e.g., verify that $u_x + u_y = u_z$ for binary numbers), stack-based tasks (e.g., verify that a sequence of brackets is properly balanced), grammar recognition tasks (e.g., verify that a string matches a regular pattern), and simpler control tasks (parity, even-pairs, majority). Each task requires a specific algorithm, usually a deterministic finite automaton for regular languages, or a specific computation involving a stack, counter, or content buffer. The original FLaRe-methodology trains each of three architectures (RNN, LSTM, Transformer) across four loss objectives (recognition-only, and combinations with auxiliary language-modeling and next-symbol prediction heads), selecting the best configuration per architecture per task for reported results. We add and train a fourth architecture, the Mamba SSM, in most of our experiments.

Activation Patching

To determine whether a trained model causally relies on a specific feature F for its classification decisions, we adapt causal activation patching (Meng et al., 2022; Wang et al., 2023) as follows. Consider a trained model M that maps input string s to a scalar logit $M(s) \in \mathbb{R}$ (positive value $\rightarrow$ predict positive; negative value $\rightarrow$ predict negative). For a pair of inputs ($s_{\text{clean}}, s_{\text{corrupt}}$) that differ only along feature F , we perform three forward passes: Clean pass: run M on s_{clean} , record logit $L_{\text{clean}} = M(s_{\text{clean}})$. Also record intermediate activations at each layer. Corrupt pass: run M on s_{corrupt} , record logit $L_{\text{corrupt}} = M(s_{\text{corrupt}})$. Patched pass: run M on s_{clean} , but at a specified intermediate layer l , replace M 's activation with the activation recorded from the corrupt pass at layer l (at the same sequence position). Record the resulting logit L_{patched} . We compute two quantities per patching intervention: Behavioral logit gap = $|L_{\text{clean}} - L_{\text{patched}}|$. This measures how much the model's output changes when we replace the intermediate representation of the patched feature. A large gap indicates the patched feature is causally load-bearing for the classification decision at that position and layer. Restored fraction = $(L_{\text{patched}} - L_{\text{corrupt}}) / (L_{\text{clean}} - L_{\text{corrupt}})$. This measures how much of the clean-to-corrupt logit difference is restored by patching. A value near 1.0 indicates the patched feature does not contribute to the classification decision. We define causally effective as behavioral logit gap ≥ 0.05 , based on empirical calibration against null intervention controls. For each candidate feature F and each task, we programmatically construct 25 counterfactual pairs. Each pair consists of a genuine input from the model's train/test distribution and a minimallymodified variant that differs only along the target feature F .

FLaRe Negative Sampling

50% of the negative examples are uniformly random, while the other 50% are K-edit perturbed negatives. The K-edit process is as follows: Sample a genuine positive example. Sample the number of edits K from a geometric distribution with success rate 0.5 (heavily favoring small K). Apply K edits, each chosen uniformly from {INSERT, REPLACE, DELETE} and applied at a uniformly-random position. INSERT adds a random character from the alphabet; REPLACE substitutes a random character in the string with a random character from; DELETE removes one random character. Discard if the result is still a positive example.

Experiments

We first replicate the FLaRe paper's results on our own device, with the addition of the Mamba architecture. Initial exploration of the results led us to perform causal activation patching analysis on Parity, Majority, and Even-Pairs. When our causal patching control tests failed on Marked-Reversal, we suspected that FLaRe benchmark had some shortcut vulnerabilities. We did three-condition causal patching on 13 identified shortcut-vulnerable tasks: structural feature patching, shortcut feature patching, and target computation patching. We found that one of these 13 languages had developed a valid computation directly relevant to its task: cycle-navigation. We thus excluded this language from the remaining experiments. Our reconstructed dataset kept 75% of the original FLaRe dataset, only replacing the 50% of negative samples that were generated by the K-edit procedure. We carefully constructed a negative dataset specific to each language, that required the target computation in order to reject. We evaluated the original FLaRe-trained models on this new test set. We also used this data to train new models and again, evaluate on this new test set. Post this experiment, we performed causal activation patching on Dyck-2-3 and Repeat-01 to confirm target computation. After this, we transitioned to a new evaluation method: Next-Character-Prediction. This was first introduced in a Bhattamishra et al. (2024) paper. In NCP, the model receives a prefix of a valid string and at each position i , outputs the probability distribution over the alphabet of which characters are valid continuations at position $i + 1$. We used NCP for 7 tasks on the Transformer model, with 5 seeds for each task, training data length 10-100, and test data length 100. The Bhattamishra paper uses independent, identically distributed uniform sampling, which creates sequences with highly predictable empirical frequencies. We thus propose distribution-diverse training, and we report results on both i.i.d. and distribution-diverse training. We measure two metrics. The first is per-position accuracy: each output position independently, we ask whether the model's top-1 predicted next symbol belongs to the ground-truth valid next-symbol set and collate this value across the string. The second is whole-sequence accuracy: this answers if every position in a test example is predicted correctly. Both of these metrics are averaged over the 5,000 test examples.

Results

Task	FLaRe-trained	Corrected-trained	Task	FLaRe-trained	Corrected-trained
marked-copy	63.9	68.8	marked-copy	57.8	74.0
marked-reversal	64.2	70.3	marked-reversal	62.8	73.1
odds-first	66.0	72.5	odds-first	60.2	69.3
bucket-sort v1	73.1	92.0	bucket-sort v1	66.6	78.6
bucket-sort v2	65.3	63.9	bucket-sort v2	66.5	66.8
dyck-2-3	87.6	98.6	dyck-2-3	75.1	86.4
stack-manipulation	72.6	72.2	stack-manipulation	67.2	70.2
missing-duplicate-string	71.6	75.0	missing-duplicate-string	72.2	71.0
repeat-01	92.7	100.0	repeat-01	96.3	94.0
binary-addition	70.6	67.8	binary-addition	71.4	71.9
binary-multiplication	70.8	71.7	binary-multiplication	68.3	74.5
compute-sqrt	70.7	67.3	compute-sqrt	71.6	64.3
modular-arithmetic-simple	74.9	74.9	modular-arithmetic-simple	74.9	75.0

Table 1 & 2: RNN and LSTM FLaRe-trained vs corrected-trained on corrected test set

Task	FLaRe-trained	Corrected-trained	Task	FLaRe-trained	Corrected-trained
marked-copy	55.5	65.9	marked-copy	60.5	69.5
marked-reversal	58.0	60.5	marked-reversal	62.1	69.1
odds-first	59.0	63.9	odds-first	62.5	71.0
bucket-sort v1	58.1	72.6	bucket-sort v1	82.7	84.9
bucket-sort v2	62.3	65.4	bucket-sort v2	73.8	65.8
dyck-2-3	63.8	60.4	dyck-2-3	83.5	96.3
stack-manipulation	58.9	68.9	stack-manipulation	74.6	67.1
missing-duplicate-string	65.3	69.6	missing-duplicate-string	72.0	70.1
repeat-01	59.0	76.9	repeat-01	95.5	95.4
binary-addition	58.6	67.4	binary-addition	68.6	75.9
binary-multiplication	56.0	69.2	binary-multiplication	71.6	78.5
compute-sqrt	58.8	60.3	compute-sqrt	65.0	63.7
modular-arithmetic-simple	66.3	65.6	modular-arithmetic-simple	74.9	74.8

Table 3 & 4: Transformer and Mamba FLaRe-trained vs corrected-trained on corrected test set

Task	Best whole-seq.	Mean whole-seq.	Best per-pos.	Mean per-pos.
repeat-01	100.0%	100.0%	100.0%	100.0%
bucket-sort	97.2%	79.6%	100.0%	99.6%
marked-copy	86.0%	75.6%	98.2%	89.0%
binary-addition	3.7%	3.4%	85.7%	85.1%
stack-manipulation	0.5%	0.4%	77.5%	75.6%
marked-reversal	0.0%	0.0%	88.4%	83.3%
odds-first	0.0%	0.0%	68.1%	65.4%

Table 5: NCP performance under standard i.i.d. training. Results are evaluated on 5,000 examples at length 100.

Task	Best whole-seq.	Mean whole-seq.	Best per-pos.	Mean per-pos.
repeat-01	100.0%	100.0%	100.0%	100.0%
bucket-sort	99.7%	91.2%	100.0%	99.8%
marked-copy	89.2%	85.0%	97.7%	87.5%
stack-manipulation	37.2%	36.0%	94.4%	93.9%
marked-reversal	0.0%	0.0%	89.0%	85.3%
binary-addition	0.0%	0.0%	78.9%	78.1%
odds-first	0.0%	0.0%	69.7%	66.3%

Table 6: NCP performance under distribution-diverse training. Results are evaluated on 5,000 examples at length 100.

Language	Transformer	RNN	LSTM	Mamba
Even Pairs	1.00	0.56	0.92	0.95
Repeat 0-1	0.71	0.96	0.98	0.97
Parity	0.57	0.77	0.90	0.60
Cycle Navigation	0.85	0.93	0.90	0.88
Modular Arithmetic Simple	0.74	0.99	1.00	0.96
Dyck-(2,3)	0.71	0.96	0.90	0.92
First	0.98	0.90	0.96	0.82
Majority	0.98	0.91	0.97	0.90
Stack Manipulation	0.63	0.85	0.81	0.86
Marked Reversal	0.62	0.77	0.72	0.66
Unmarked Reversal	0.58	0.76	0.74	0.60
Marked Copy	0.61	0.73	0.66	0.64
Missing Duplicate String	0.67	0.87	0.89	0.79
Odds First	0.62	0.81	0.69	0.65
Binary Addition	0.67	0.80	0.82	0.77
Binary Multiplication	0.64	0.81	0.79	0.83
Compute Sqrt	0.63	0.80	0.83	0.70
Bucket Sort	0.67	0.82	0.69	0.80

Table 7: FLaRe paper's replicated results with the addition of the fourth architecture (Mamba)

Conclusion

The FLaRe dataset was shown to carry a severe shortcut vulnerability due to its K-edit perturbed negatives sampling strategy. Most of the FLaRe models were shown to be using shortcuts or trivial features, via activation patching. When retested on the revised dataset, most of the FLaRe-trained models fell to 65-75%, because they were unable to generalize to the 25% of the test set that required performing the target computation to reject. The 4 architectures being retrained on the revised dataset yielded mixed results. While many task accuracies improved significantly, only Dyck-2-3 and Repeat-01 reached extremely high levels of accuracy, suggesting having actually acquired the target computation in most or all of the cells. We verified this causally, using activation patching. The NCP results were very successful in proving Transformers' capabilities for Repeat-01, Bucket-Sort, Marked-Copy, and Stack-Manipulation whole-sequence accuracy. The remaining 3 languages still performed very well in the per-position metric. The distribution-diverse training did better overall than the i.i.d training.

$$\text{Acc}_{\text{pos}} = \frac{1}{T} \sum_{i=1}^T \mathbf{1} [\hat{y}_i \in Y_i] \quad \text{Acc}_{\text{whole}} = \mathbf{1} [\forall i, \hat{y}_i \in Y_i]$$

References

Satwik Bhattamishra, Michael Hahn, Phil Blunsom, and Varun Kanade. *Separations in the representational capabilities of transformers and recurrent architectures*. In Advances in Neural Information Processing Systems, 2024.

Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. *Training neural networks as recognizers of formal languages*. In International Conference on Learning Representations, 2025.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. *Neural networks and the chomsky hierarchy*. In International Conference on Learning Representations, 2023.

Acknowledgements

We acknowledge the FLaRe paper's novelty in creating a sampling strategy that can be generalized to any formal language task. We do not discredit their work at all. Thank you to our professor, Dr. Junyi Tu, our PI, Dr. Enyue Lu, and the rest of our NSF EXERCISE REU colleagues.