

The Long and Short of It: An experimental investigation into the role of shorter strings in learning regular languages

Logan Swanson

LOGAN.SWANSON@STONYBROOK.EDU

Department of Linguistics

Institute for Advanced Computational Science

Stony Brook University

Sarah Payne

SARAH.PAYNE@STONYBROOK.EDU

Department of Linguistics

Institute for Advanced Computational Science

Stony Brook University

Adil Soubki

ASOUBKI@CS.STONYBROOK.EDU

Department of Computer Science

Institute for Advanced Computational Science

Stony Brook University

Sam van der Poel

SAMVANDERPOEL@GATECH.EDU

Department of Computer Science

Georgia Tech University

Jeffrey Heinz

JEFFREY.HEINZ@STONYBROOK.EDU

Department of Linguistics

Institute for Advanced Computational Science

Stony Brook University

Abstract

We test the performance of state merging algorithms against four neural network architectures on MLREGTEST, a benchmark for learning regular languages from positive and negative data (van der Poel et al., 2024). Prior work by Soubki and Heinz (2023) on a preliminary version of MLREGTEST found that neural learning models generally outperformed state merging algorithms on this benchmark. However, Soubki and Heinz (2023) also note that MLREGTEST excludes strings shorter than length 20 and report a follow-up experiment demonstrating that the addition of shorter strings dramatically improves the performance of state merging algorithms. They did not examine the performance of neural models with shorter strings.

We expand on this work by conducting several experiments using the published version of MLREGTEST. Firstly, we conduct a gridsearch over the hyperparameters provided by the FLEXFRINGE software (Verwer and Hammerschmidt, 2017, 2025) to identify the best-performing state merging system. We then report the results of this state merging algorithm on experimental setups which vary by training set size, inclusion of shorter training strings, and inclusion of intermediate-length training strings. We also subject the neural baselines from van der Poel et al. (2024) to these same experiments, and compare the performance between these and the state merging algorithm.

We conclude that the gated recurrent unit has the strongest performance across the benchmarks, with some interesting caveats. Namely, the state merging algorithm outperforms all neural architectures on regular languages which count modulo some positive integer. Additionally, the state merging algorithm sees much greater improvement and more

resiliency to adversarial test sets from the inclusion of shorter strings, while the neural models do much better with longer strings and larger data samples. These results suggest that state merging approaches may be more attuned to learning from shorter, smaller input data, while neural learners need the opposite.

Keywords: state-merging, regular languages, finite-state automata, neural networks

1. Introduction

MLREGTEST (van der Poel et al., 2024) is a benchmark for learning regular languages from positive and negative data. Using this benchmark, this paper compares the learning behavior of neural and state-merging learning algorithms on the regular languages, in particular exploring the role of string length in the algorithms’ performance.

The performance of the state-merging algorithms RPNI, EDSM, and ALERGIA was examined by Soubki and Heinz (2023) on a preliminary version of MLREGTEST and found to be generally weaker than that of the neural learning models investigated by van der Poel et al. (2024). However, since MLREGTEST excludes strings of length 19 or shorter in order to achieve a balance of positive and negative strings in its datasets, Soubki and Heinz (2023) also report follow-up experiments which included shorter training strings and show that they dramatically improve the performance of the state-merging algorithms. This indicates that string length may be an important factor in learning behavior for these algorithms, and opens the question of whether neural learning approaches are similarly impacted.

In this context, this paper makes the following contributions. First, we examine the performance of state merging algorithms on the final version of MLREGTEST, not the preliminary version. Second, instead of exclusively running RPNI and EDSM out of the box, we conduct a limited gridsearch over the parameters provided by the flexfringe software (Verwer and Hammerschmidt, 2017, 2025) to find suitable hyperparameters for merging states. Third, we report the results of this resulting state-merging algorithm on five experimental setups: short strings alone (ONLYSHORT), the small-sized dataset of intermediate-length strings (SMALL ONLYINT), the small-sized dataset plus the short strings (SMALL PLUSSHORT), the medium-sized dataset of intermediate-length strings (MEDIUM ONLYINT), and the medium-sized dataset plus the short strings (MEDIUM PLUSSHORT). Finally, we subject the neural baselines from van der Poel et al. (2024) to these same experiments for comparison.

In terms of overall accuracy, we find that the gated recurrent unit generally outperforms all other algorithms tested (both neural and state-merging) across experiments. Interestingly, however, the state-merging algorithm outperforms the neural methods on regular languages that count modulo some positive integer. Additionally, the state-merging algorithm benefits much more from the addition of short strings than the neural models do—both in terms of performance and in terms of resilience to adversarial testing data. The neural approaches, meanwhile, are far more sensitive to longer strings and see only moderate improvement with the addition of short ones.

The remainder of this paper is organized as follows. Section 2 introduces the learning algorithms that we examine, both the state-merging (§2.1) and neural (§2.2) systems. Section 3 summarizes the setup of the MLREGTEST datasets, explains our addition of short strings to these datasets, and outlines our full experimental paradigm. Section 4 reports

our findings, examining the effects of string length (§4.1), type of test set (§4.2), specific language class (§4.3), and the logical levels and ordering relations those classes are built on (§4.4). Finally, Section 5 offers ideas for future directions and concludes.

2. Learning Systems

2.1 State-Merging Systems

Following Soubki and Heinz (2023), we use the state-merging software FLEXFRINGE (Verwer and Hammerschmidt, 2025),¹ which implements a range of state-merging strategies using the red/blue framework (see de la Higuera (2010)). Generally speaking, these algorithms take as input a list of strings each labeled as to whether or not they belong in the target language. They first construct a prefix tree from this input, with accepting and non-accepting states notated accordingly. Then, states in the prefix tree are merged together until finally, a finite state acceptor is returned which represents the learned grammar. In general, pairs of states are scored for their “mergability”, and states with sufficient scores are merged in a particular order as determined by the specific hyperparameter settings being used. These different hyperparameter settings correspond to different individual state-merging algorithms. For example, the EDSM (Lang et al., 1998) and RPNI (Oncina et al., 1992) algorithms are particular strategies among the many which FLEXFRINGE can instantiate. FLEXFRINGE also can also learn stochastic regular languages with state-merging algorithms such as ALERGIA (Carrasco and Oncina, 1994, 1999). However, we exclude ALERGIA from consideration because of its comparatively low performance noted by Soubki and Heinz.

FLEXFRINGE provides several parameters which determine its state-merging strategy. To find the best model, we conducted a limited gridsearch over the following subset: **PerformFirst** (EDSM vs. RPNI), **Extend** (color blue states red that cannot be merged with any red), **ShallowFirst** (consider coloring blue states closest to the root first), and **SearchStrategy** (searchdeep, searchlocal, searchglobal, searchpartial, or none). Grid searching was done on the small partition of the PLUSSHORT data using overall accuracy on the LARGE development set (these data sets are described in §3). The best performing model, used in the remainder of the paper, was **PerformFirst** = 0 (i.e., RPNI), **Extend** = 1, **ShallowFirst** = 0, and **SearchStrategy** = **SearchDeep**. These settings for **ShallowFirst** and **SearchStrategy** represent a departure from the default settings used by Soubki and Heinz (2023), which were **ShallowFirst** = 1, and **SearchStrategy** = **none**.

The SearchDeep strategy deserves some discussion. This strategy considers the tree of all possible sequences of merges and the sizes of the resulting finite state acceptors. It ultimately returns the acceptor with the fewest number of states. Given that the tree representing all possible sequences of merges is quite large (and grows larger with the size of the initial prefix tree), it is time-consuming to run. Notably, without this parameter setting, the FLEXFRINGE models typically run in a matter of seconds (compared to minutes or hours for the neural models).

We held several other parameters at their default state because modifying them led to errors in the **FlexFringe** executable. These are: **ReverseTraces**, **DepthCheck**, **SymbolCheck**,

1. We use the same version of FLEXFRINGE used by Soubki and Heinz: commit 4d1449c of the software available here.

SinksOn, **SinkCount**, and **MergeScore**. However, we have since learned more about these parameters (Sicco Verweer, p.c.), and we believe these errors can be avoided in future work (see Section 5 for a discussion of future research directions).

2.2 Neural Systems

Following van der Poel et al. (2024), we test four neural architectures, all consisting of a trainable embedding layer followed by a unidirectional recurrent module (either a **SIMPLE** RNN, a **GRU**, or an **LSTM**) or two consecutive **TRANSFORMER** blocks, dense feed-forward layers, dropout, layer normalization, and softmax activation. We trained these neural models on the same datasets as the FLEXFRINGE models (i.e., lists of strings labeled according to membership in the target language), and provided the development sets for validation. All neural networks use the same hyperparameters as van der Poel et al.², which were identified using a grid search to find the best-performing settings.

3. Experimental Setup

MLREGTEST (van der Poel et al., 2024) contains training, development, and test data from 1,800 regular languages. The languages were organized along three major dimensions: alphabet size (**Alph**), logical complexity, and model-theoretic representations of the ordering relation in strings. The logical complexity and ordering relations gave rise to 16 distinct classes, shown in Table 1 (reproduced from Soubki and Heinz (2023)).

Five logics are considered: Monadic Second Order (MSO), First Order (FO), Propositional (Prop), Conjunctions of Negative literals (CNL) and Disjunctions of Positive Literals (DPL). CNL and DPL are complements of the other and both are properly contained within Prop, which is properly contained in FO, which is contained within MSO. The three model-theoretic ordering relations are successor ($\triangleleft$), tier-successor ($\triangleleft_T$), and precedence ($<$), which correspond to the concepts of substring, tier-substring, and subsequence, respectively. We use the term *factor* to refer to these concepts, and languages in MLREGTEST vary the sizes of these factors. A tier T is a subset of the alphabet which can lead to a change of state. Non-tier elements are neutral and can be freely inserted or deleted (Lambert, 2023). MLREGTEST considers an additional subclass of purely periodic languages they call Z_p . For example, the language whose words contain an even number of a symbols belongs to Z_p . Three additional parameters further describe the languages within each class. These are **Tier** (the size of the tier), **k** (the size of the factors), **j** (the size of the threshold), and **i** (an index uniquely specifying the language within the class and alphabet size, ranging from 0 to 9).

The names of these classes (**Class**) are given in Table 1, alongside their logical characterization. These classes and their relationships are discussed by many, including McNaughton and Papert (1971); Thomas (1982, 1997); Rogers and Pullum (2011); Rogers et al. (2013); Rogers and Lambert (2019). See Lambert et al. (2021) for an approach to learning these classes individually. Readers are referred to van der Poel et al. (2024) for additional details regarding these language classes.

2. The implementations of the neural networks are available at <https://github.com/heinz-jeffrey/subregular-learning/>

Logic	Representation	Language Name	Abbreviation
MSO	$\triangleleft$	Regular	Reg
	$\triangleleft_T$		
	$<$		
FO	$\triangleleft$	Locally Threshold Testable	LTT
	$\triangleleft_T$	Tier Locally Threshold Testable	TLTT
	$<$	Star-Free	SF
Prop	$\triangleleft_T, <$	Tier Piecewise Locally Testable	TPLT
	$\triangleleft, <$	Piecewise Locally Testable	PLT
	$\triangleleft_T$	Tier Locally Testable	TLT
	$\triangleleft$	Locally Testable	LT
	$<$	Piecewise Testable	PT
CNL	$\triangleleft$	Strictly Local	SL
	$\triangleleft_T$	Tier Strictly Local	TSL
	$<$	Strictly Piecewise	SP
DPL	$\triangleleft$	Complement of Strictly Local	coSL
	$\triangleleft_T$	Complement of Tier Strictly Local	TcoSL
	$<$	Complement of Strictly Piecewise	coSP

Table 1: Logical characterizations and names of classes in MLREGTEST.

Van der Poel et al. (2024) use the Language Toolkit (Lambert, 2024) to verify that every language in some class A in MLREGTEST belongs to A and does not belong to any other class B, where B is a subclass of A or is incomparable with A.

The datasets for each language come in three sizes: SMALL (1,000 strings), MEDIUM (10,000 strings), and LARGE (100,000 strings). For each language, the datasets are nested; SMALL is contained in MEDIUM, which is contained in LARGE. Each dataset includes equal numbers of positive and negative strings.

The MLREGTEST training sets contain strings of lengths 20-29, sampled with replacement (these sets are referred to here as INTERMEDIATE, to distinguish them from sets of shorter or longer strings). The development sets also contain strings of lengths 20-29, sampled without replacement in a manner to keep the development sets disjoint from the training set.

Soubki and Heinz found that augmenting the training set with strings of length 1-19 significantly improved FLEXFRINGE performance. Soubki and Heinz built a training set of short strings for each language as follows. For each string length (1-19), 25 positive and 25 negative strings were sampled with replacement. If no positive or negative strings of a given length existed, none were drawn for that length. This method resulted in a supplemental SHORT training set for each language, containing up to 1,000 shorter strings. It is worth noting that a key difference between these SHORT sets and the INTERMEDIATE sets originally included in MLREGTEST is that for many regular languages, there is no guarantee that 25

positive (negative) strings can be generated for a given length. For example, many regular languages contain *no* strings of length 1 or 2.

This has three important implications for the nature of the SHORT training sets. Firstly, most SHORT training sets contain significantly less than 1,000 examples, and the number of examples is not standardized across languages. Second, due to the small numbers of positive (negative) strings for certain lengths and the procedure of sampling with replacement, specific strings are more likely to be repeated several times in the SHORT training sets. Finally, the SHORT training sets are not balanced between numbers of positive and negative strings. Some contain mostly positive examples, and some mostly negative. The original MLREGTEST excluded shorter string lengths in order to run controlled experiments with balanced training sets. This work, however, is intended to explicitly examine the impact of shorter strings on the learning of regular languages, and we therefore consider the SHORT training data to be valuable.

MLREGTEST contains four test sets (`TestType`) per language, based on the string length and the method of generation. INTERMEDIATE test sets have strings of length 20-29, matching the training data, while LONG sets have strings of length 31-50. RANDOM test sets were created by sampling without replacement in a manner to keep them disjoint from the corresponding training and development sets. ADVERSARIAL sets sampled pairs of positive and negative examples with an edit distance of 1. Like the RANDOM test sets, these were also sampled in a manner to keep them disjoint from the training and development sets. (Of course for the LONG test sets nothing needed to be done to ensure they were disjoint from the training and development sets, since these only included strings of shorter lengths.) Henceforth, we refer to these test sets as IR, LR, IA, LA for INTERMEDIATE RANDOM, LONG RANDOM, INTERMEDIATE ADVERSARIAL, and LONG ADVERSARIAL, respectively.³

Following van der Poel et al. (2024) and Soubki and Heinz (2023), we evaluate trained models only on the LARGE test sets (recall that the sets are nested, so the LARGE test sets contain the MEDIUM and SMALL ones). Note that these test sets are limited to strings of INTERMEDIATE or LONG length, with SHORT strings absent from testing. This means that the undesirable properties of the SHORT sets described above are limited to our training regime, and do not impact our testing setup (in particular, our test sets maintain their balance between positive and negative examples). The purpose of including SHORT strings in training is to investigate how much these strings help models to generalize, and the INTERMEDIATE and LONG testing data are adequate to evaluate this.

In their preliminary experiments, Soubki and Heinz used an early, unfinished version of MLREGTEST which included issues with string sampling (see footnote 5 on page 19 of van der Poel et al. (2024)) as well as some errors in the underlying automata for generating certain languages. Additionally, Soubki and Heinz tested only three state merging variants (corresponding to the RPNI, EDSM, and ALERGIA algorithms) by setting the parameters of the FLEXFRINGE model to fixed default settings. In this work, we replicate their analyses on the final, published version of MLREGTEST, as well as conducting a gridsearch to identify

3. In van der Poel et al. (2024), these were called Short Random (SR), Long Random (LR), Short Adversarial (SA), and Long Adversarial (LA), respectively.

the best-performing FLEXFRINGE parameter settings.⁴ We also extend their experiments to provide a comparison against neural network models on the SHORT datasets.

We conducted three kinds of experiments. In the ONLYSHORT experiments, only the short training set was used for training. In the PLUSSHORT experiments, the training set was made up of the original MLREGTEST training sets augmented with the short training set. In the ONLYINT experiments, the original MLREGTEST training sets were used (without the short training set).

We excluded languages with alphabet sizes of 64, thus considering 1,140, in order to compare our results with Soubki and Heinz (2023). This submission also excludes the Large training sets because the SearchDeep strategy caused FLEXFRINGE to run too long during training (see §5 for notes on future work in this direction).

Thus for each language, for each learning algorithm, we considered five training regimes (**TrainType**): ONLYSHORT, SMALL PLUSSHORT, SMALL ONLYINT, MEDIUM PLUSSHORT, and MEDIUM ONLYINT. Each trained model (**AlgoType**) was tested on the aforementioned four test sets.

Note that we have now introduced terminology to categorize data sets according to two properties: the *total number* of strings they contain, and the *lengths* of these strings. The terms SMALL, MEDIUM, and LARGE refer to the number of examples in a dataset (1,000, 10,000, and 100,000 examples respectively). Meanwhile, the terms SHORT, INTERMEDIATE, and LONG refer to the lengths of the strings in a set (1-19, 20-29, and 31-50 characters respectively).

The PLUSSHORT and ONLYINT experiments parallel the experiments in Soubki and Heinz (2023) on a preliminary version of MLREGTEST, with the additions of using FLEXFRINGE parameter settings established by a gridsearch (see §2.1) and including neural models in the experimental setup. The ONLYSHORT experiments for both FLEXFRINGE and the neural models are novel to this paper and help clarify the role short training strings play for learning regular languages.

4. Results

Given the balanced nature of the tests sets, very strong correlations were found across all performance measures (accuracy, brier score, f1). Therefore, we report aggregate accuracy as the primary measurement of performance for all models.

We use the same non-parametric methods and full factorial design as van der Poel et al. (2024) and Soubki and Heinz (2023) in our analysis, as advocated by Demšar (2006) and Stapor (2018). The factorial design mean-aggregates the factors **Tier**, **k**, **j**, and **i**. As such, the value of each cell of the design is the average of accuracy scores of observations that have the same values of **Alph**, **Class**, **TrainType**, **AlgoType**, and **TestType** but different values of **Tier**, **k**, **j**, and **i**. Thus, any of the five factors **Alph**, **Class**, **TrainType**, **TestType**, or **AlgoType** are treatment variables and can be tested for their association with accuracy using the non-parametric advocated for above.

4. MLRegTest is publicly available with Dryad <https://doi.org/10.5061/dryad.dncjsxm4h> under the license *CC0 1.0 Universal (CC0 1.0) Public Domain Dedication* (<https://creativecommons.org/publicdomain/zero/1.0/>).

MODEL	ONLY SHORT	SMALL PLUSSHORT	SMALL ONLYINT	MEDIUM PLUSSHORT	MEDIUM ONLYINT
FlexFringe	0.762	0.784	0.557	0.808	0.592
Simple	0.671	0.733	0.714	0.785	0.780
LSTM	0.649	0.730	0.685	0.866	0.856
Transformer	0.712	0.770	0.758	0.822	0.815
GRU	0.793	0.912	0.837	0.947	0.940

Table 2: Mean accuracy score across all test sets for all languages by experiment type and model.

The Friedman rank sum test is used to determine if the factor being held for treatment contributes significantly to predicting model accuracy by holding the null hypothesis that all variations of the value contribute equally. Additional analysis using the Nemenyi-Wilcoxon-Wilcox all-pairs test identified which pairs differ significantly.

4.1 Effect of Short Training Strings

Table 2 gives the mean accuracy score of each model across languages, experiments, and test sets. The Friedman rank sum test indicated that the training condition was significant for every model ($p < 2.2\text{e-}16$). The results in the PLUSSHORT and ONLYINT experiments are consistent with Soubki and Heinz (2023) and van der Poel et al. (2024), respectively.

Under the hyperparameters used here, the GRU consistently outperforms the other models across all experiments. Interestingly, FLEXFRINGE outperforms the other neural models in the ONLYSHORT and SMALL PLUSSHORT experiments. In all other training conditions, the neural networks outperform FLEXFRINGE, with the exception of the Simple RNN in the MEDIUM PLUSSHORT condition.

Another important observation is that the addition of the short training data makes a major impact on the performance of the FLEXFRINGE model. Comparing the ONLYINT to the PLUSSHORT training conditions, the FLEXFRINGE model jumps at least 20 percentage points in each case. The neural models, on the other hand, are much less sensitive to the inclusion of the short training data, with the largest improvement being about 4.5 points (for the LSTM model in the SMALL training regime). These differences persist across test sets as shown in Figure 1.

Conversely, the neural networks see a much bigger jump in performance from the addition of intermediate-length strings. Comparing the ONLYSHORT to the SMALL PLUSSHORT training regimes, each neural model jumps at least 5 percentage points, and GRU jumps over 10. By comparison, the FLEXFRINGE model only improves by 2.2 percentage points.

Additionally, there is a larger jump in performance for the neural models from the SMALL to MEDIUM training conditions than there is for FLEXFRINGE. From SMALL PLUSSHORT to MEDIUM PLUSSHORT, FLEXFRINGE’s performance increases by about only 2 points, whereas the neural networks each see a performance increase of about 5 points, except for the LSTM which shows a nearly 15 point increase.

Taken together, these results suggest that FLEXFRINGE benefits far more from the inclusion of short strings, while the neural models rely more on long strings to achieve their

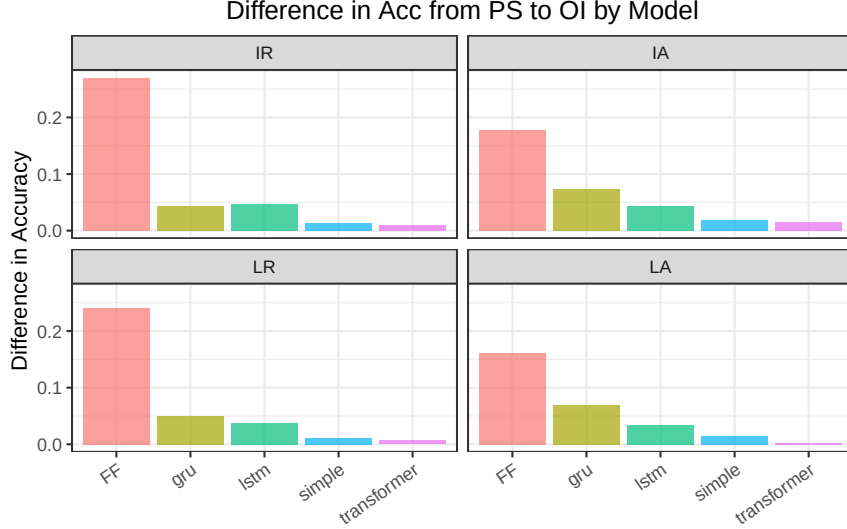


Figure 1: Difference in overall accuracy between SMALL PLUSSHORT and SMALL ONLYINT by model and test type. ⁵

Training condition	IR	IA	Δ error rate	LR	LA	Δ error rate
ONLYSHORT	0.832	0.725	$\times 1.64$	0.791	0.698	$\times 1.44$
SMALL PLUSSHORT	0.845	0.749	$\times 1.62$	0.815	0.727	$\times 1.48$
SMALL ONLYINT	0.576	0.544	$\times 1.08$	0.569	0.539	$\times 1.07$
MEDIUM PLUSSHORT	0.862	0.774	$\times 1.64$	0.840	0.755	$\times 1.53$
MEDIUM ONLYINT	0.613	0.579	$\times 1.09$	0.603	0.571	$\times 1.08$

Table 3: Mean FLEXFRINGE accuracy by test type and training condition

performance. This seems to be particularly true for the GRU and LSTM which see the large performance increases in the SMALL and MEDIUM conditions.

4.2 FlexFringe Performance by Test Set

Van der Poel et al. (2024) found that the performance of the neural models varied significantly depending on the test sets. In particular, the accuracy on the adversarial test sets was much lower as compared to the random test sets. The Friedman rank sum test for the FLEXFRINGE model tested here is also significant ($p < 2.2e-16$), with the biggest factor again being the adversariality of the test sets (Table 3).

Interestingly, the state merging algorithm demonstrated less of a drop in performance between the random and adversarial conditions than the neural models. Table 3 reports the mean accuracies in these test sets by training condition.

Table 3 shows that, even in the ONLYSHORT training condition, the performance drop is at most roughly 10-11 points. This is in contrast to the neural models, which shows at best about a 15 point drop in accuracy in the ONLYSHORT condition (the GRU; see Table 4) and at worst about a 20 point drop in accuracy (see Tables tables 8 to 10 in the appendix). In the

Training condition	IR	IA	Δ error rate	LR	LA	Δ error rate
ONLYSHORT	0.882	0.732	$\times 2.27$	0.850	0.708	$\times 1.95$
SMALL PLUSSHORT	0.971	0.867	$\times 4.59$	0.961	0.851	$\times 3.82$
SMALL ONLYINT	0.911	0.774	$\times 2.54$	0.899	0.762	$\times 2.36$
MEDIUM PLUSSHORT	0.975	0.924	$\times 3.04$	0.971	0.916	$\times 2.90$
MEDIUM ONLYINT	0.972	0.913	$\times 3.11$	0.970	0.905	$\times 3.17$

Table 4: Mean GRU accuracy by test type and training condition

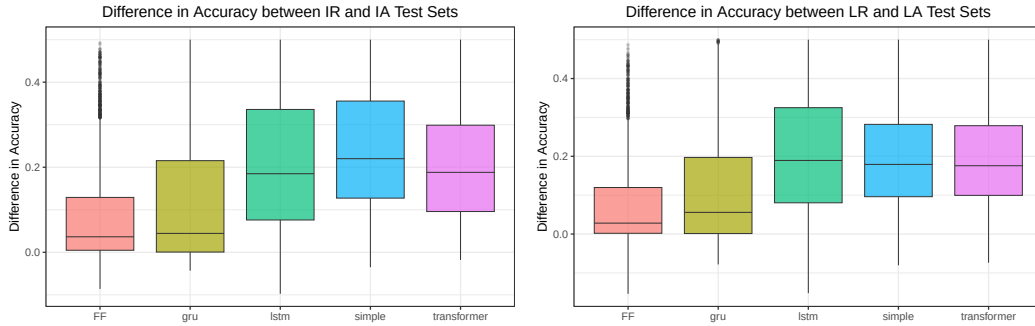


Figure 2: Difference in overall accuracy between random and adversarial test sets by model and test string length.

MEDIUM PLUSSHORT condition, the GRU does a better job with the adversarial set in terms of absolute difference in accuracy, dropping about 5 percentage points where FLEXFRINGE drops about 9. On the other hand, that 9 point drop represents only a 1.6-factor increase in the number of errors made by FLEXFRINGE, while errors made by the GRU model increase by a factor of 3.

To investigate whether the drops between FlexFringe and each neural model were statistically significant, we conducted statistical analyses using the paired t-test. Pairing was done by matching accuracy scores for FlexFringe and a neural model according to the factors `Alph`, `Class`, `Tier`, `k`, `j`, and `i`. The differences between accuracy in the IR and IA test sets, and the differences between LR and LA test sets were examined separately. Since there were four neural models and 2 conditions (INTERMEDIATE and LONG), we ran 8 paired t-tests. Every test came back significant ($p < 2.2e-16$). Figure 2 shows the differences in accuracy between the test sets for each of the models.⁶

In general, the paired t-test should be used for normally distributed data when the numbers of data points are small. As the number of data points increases, the normal population assumption becomes less important because of the Central Limit Theorem (Agresti, 2018, pp. 119, 192-3). Here, the histograms did not appear normally distribution, but there were over a thousand data points being analyzed, so we concluded the paired t-test was appropriate.

6. This figure is computed directly from the full table of experimental observations and not from the mean-aggregate cells in the factorial design.

Accuracy	Class	Accuracy	Class	Accuracy	Class	Accuracy	Class
0.865	Zp	0.721	Reg	0.686	TcoSL	0.652	TLT
0.761	SF	0.718	PLT	0.686	TSL	0.652	PT
0.753	SL	0.713	LT	0.665	TPLT	0.617	SP
0.749	coSL	0.696	LTT	0.656	TLTT	0.615	coSP

Table 5: Aggregate mean accuracy over all training conditions and test sets of FLEXFRINGE in decreasing order by language **Class**.

Accuracy	Class	Accuracy	Class	Accuracy	Class	Accuracy	Class
0.915	Zp	0.810	SF	0.764	TcoSL	0.712	TPLT
0.859	SL	0.808	LT	0.763	TSL	0.651	PT
0.846	coSL	0.782	LTT	0.719	TLTT	0.628	SP
0.811	PLT	0.780	Reg	0.716	TLT	0.621	coSP

Table 6: Aggregate mean accuracy over all test sets in the ONLYSHORT training condition of FLEXFRINGE in decreasing order by language **Class**.

In sum, when only short strings are provided, FLEXFRINGE is more resilient to adversarial test sets than the neural models. With longer strings and additional training data, the neural models, and especially the GRU, become more resilient to adversarial tests.

4.3 FlexFringe Performance by Language Class

The Friedman test shows that the language class is significant factor for accuracy for the FlexFringe model tested here ($p < 2.2\text{e-}16$). Table 5 shows mean accuracy for the FLEXFRINGE model aggregated across training conditions and test sets.

Examination of the post-hoc Nemenyi-Wilcoxon-Wilcox all-pairs test, however, shows that most pairwise comparisons are not significant. Significant differences generally only occur between the highest performing classes (Zp, SF, SL, coSL) and the lowest ones (SP, coSP).

However, it is striking how well the FLEXFRINGE model performs on the Zp class, which contains languages that count modulo some natural number n . The FLEXFRINGE model achieves a mean accuracy of 0.865 on this class, while the highest performing neural model (the GRU) obtains a mean accuracy of only 0.741.

To understand how the state merging algorithm performs on exclusively short strings, Table 6 shows accuracy by language class on exclusively the ONLYSHORT training condition. While the languages are not ordered identically in terms of performance as in the aggregate case, it is interesting to observe some trends. First, the extremes have much in common: Zp, SL, coSL are in the top four and SP, coSP, and PT are in the bottom four. Second, the mean accuracy is generally higher in Table 6 than in Table 5. This is likely due to the comparatively poor performance of the FLEXFRINGE model on the ONLYINT training regimes, which are included in Table 5 but not Table 6.

Abbrev.	Order Relation	Lang. Classes
SUCC	Successor	SL, coSL, LT, LTT
PREC	Precedence	coSP, PT, SF, SP
TSUCC	Tier Successor	TcoSL, TLT, TLTT, TSL
OTHER	Other	PLT, TPLT, Reg, Zp

Table 7: Language classes grouped by order relation.

These trends can be compared to those of the neural models as far as which languages they perform best on. In general, the neural learners perform best on the SF, PLT, PT, and SL classes—all of which are in the top five best languages for the state merging algorithm as well, with the notable exception of PT, which is in the bottom three. On the other hand, the neural models tend to struggle most with SP, coSP, Reg, and Zp. SP and coSP are also difficult for the FLEXFRINGE learner, but it does better (compared to other language classes) on Reg, and much better on Zp. This alignment between the state merging and neural approaches may suggest that the SF, PLT, and SL classes are intrinsically easier to learn, while the SP and coSP classes are more difficult—at least for algorithms which target the entire Reg class. Zp, Reg, and PT, meanwhile may be easier to learn by some approaches and harder by others.

4.4 FlexFringe Performance by Logical Level and Order Relation

We also checked to see whether the logical level or the ordering relation was a significant factor. The languages are grouped by the logical level as shown in Table 1, with the addition that Zp is grouped together with Reg. The Friedman rank sum test is not significant (Friedman chi-squared = 6.95, df = 4, p-value = 0.1386). To examine the ordering relation, we grouped the languages into four categories as shown in Table 7.

The Friedman test is insignificant but marginally so (Friedman chi-squared = 12.675, df = 3, p-value = 0.005395). We follow van der Poel et al. (2024), who cite Benjamin et al. (2018) in setting the significance threshold to 0.005. Due to its closeness to the significance threshold, we also report Nemenyi-Wilcoxon-Wilcox all-pairs test (Table 11 in the appendix). It indicates that this marginal significance is due entirely to the pairwise comparison of SUCC with PREC and TSUCC.

To sum up, the FLEXFRINGE model does not especially appear to be sensitive to the logical level or the ordering relation. This is not unexpected given that the state-merging algorithms it is based (e.g. RPNI and EDSM) on are intended to learn the class of deterministic finite-state automata.

5. Conclusions

Across partitions, neural models—in particular the GRU—were able to outperform FLEXFRINGE in terms of total accuracy. Strikingly, however, the state merging algorithm benefited far more from the addition of short strings than any of the neural models, seeing major jumps in performance between the ONLYINT and PLUSSHORT conditions. In fact, the state merging algorithm performed most competitively in the ONLYSHORT and SMALL PLUSSHORT con-

ditions, where it outperformed all neural models except the GRU. Additionally, the state merging algorithm showed particularly impressive performance on the Z_p class of languages (characterized by modulo counting), where it outperformed all other models. The neural models, meanwhile, were much more sensitive to the size of the training set, seeing much greater improvement between the SMALL and MEDIUM training sizes than the state merging algorithm.

These results indicate an attunement of the state merging approach to certain kinds of input samples: namely, small data and short strings. The neural methods, meanwhile, are more suited to large datasets and long strings. This finding is interesting because, in some settings, overall accuracy is not the only important factor when evaluating what makes a good learning model. In terms of modeling how humans learn natural language, for example, leveraging small data and short strings is a crucial property. Modulo counting, on the other hand, is generally considered to be outside the domain of human language patterns, so the success of the state-merging algorithm on the Z_p languages may actually suggest it is in some ways *too* powerful to be a good model for human language learning. State-merging algorithms which target aperiodic regular languages may therefore be of particular interest. Overall, the results of these experiments allow us to paint a more nuanced picture of which properties are relevant for learning regular languages and how different algorithms respond to different learning setups.

Future research can continue to expand and refine these findings in several ways. Firstly, as noted in Section 2.1, the gridsearch we conducted was limited to a small handful of the hyperparameters provided by FLEXFRINGE. Newer versions of FLEXFRINGE make it possible to vary more of these parameters and run a more complete gridsearch. This will offer further insight about what state merging strategies work best for regular language learning, as well as potentially yielding a better-performing state merging algorithm. Relatedly, while the results of our gridsearch suggest that the **SearchDeep** search strategy may offer the best performance, this strategy is too time-consuming to use at a large scale without in some way limiting its search. As explained in Section 2.1, the **SearchDeep** strategy is not just performing merges in a single prefix tree, but actually constructing a tree of all possible merges to find the smallest resulting finite state acceptor. For the LARGE data, this quickly becomes impractical. To maintain the advantages of this strategy and still allow for scalability, this search of possible merges can be terminated after a specific time interval. Doing this will allow these experiments to be expanded to the LARGE data sizes. Finally, we follow Soubki and Heinz (2023) in excluding languages with the largest alphabet size (64 characters) from our experiments, but future work should include these as well.

Acknowledgments and Disclosure of Funding

Data set creation, model training, and evaluation were completed on the Stony Brook Sea-Wulf HPC cluster maintained by Research Computing and Cyberinfrastructure, and the Institute for Advanced Computational Science at Stony Brook University and made possible by NSF grant #1531492. JH acknowledges the additional support of NSF grant #2416183. We also thank Sicco Verweir for his assistance with FlexFringe.

Appendix

Training condition	IR	IA	Δ error rate	LR	LA	Δ error rate
ONLYSHORT	0.799	0.598	$\times 2.00$	0.718	0.567	$\times 1.54$
SMALL PLUSSHORT	0.873	0.647	$\times 2.78$	0.801	0.610	$\times 1.96$
SMALL ONLYINT	0.851	0.621	$\times 2.54$	0.790	0.594	$\times 1.93$
MEDIUM PLUSSHORT	0.925	0.728	$\times 3.63$	0.828	0.659	$\times 1.98$
MEDIUM ONLYINT	0.925	0.726	$\times 3.65$	0.819	0.651	$\times 1.93$

Table 8: Mean Simple RNN accuracy by test type and training condition

Training condition	IR	IA	Δ error rate	LR	LA	Δ error rate
ONLYSHORT	0.765	0.565	$\times 1.85$	0.721	0.544	$\times 1.63$
SMALL PLUSSHORT	0.860	0.636	$\times 2.60$	0.822	0.604	$\times 2.22$
SMALL ONLYINT	0.805	0.586	$\times 2.12$	0.783	0.565	$\times 2.00$
MEDIUM PLUSSHORT	0.956	0.820	$\times 4.09$	0.923	0.767	$\times 3.03$
MEDIUM ONLYINT	0.948	0.812	$\times 3.62$	0.912	0.751	$\times 2.83$

Table 9: Mean LSTM accuracy by test type and training condition

Training condition	IR	IA	Δ error rate	LR	LA	Δ error rate
ONLYSHORT	0.846	0.649	$\times 2.28$	0.766	0.588	$\times 1.76$
SMALL PLUSSHORT	0.894	0.706	$\times 2.77$	0.836	0.646	$\times 2.16$
SMALL ONLYINT	0.879	0.684	$\times 2.61$	0.827	0.640	$\times 2.08$
MEDIUM PLUSSHORT	0.946	0.792	$\times 3.85$	0.852	0.697	$\times 2.05$
MEDIUM ONLYINT	0.942	0.786	$\times 3.69$	0.843	0.688	$\times 1.99$

Table 10: Mean Transformer accuracy by test type and training condition

	OTHER	PREC	SUCC
PREC	0.692	—	—
SUCC	0.169	0.009	—
TSUCC	0.773	0.999	0.014

Table 11: Nemenyi-Wilcoxon-Wilcon all-pairs test for the ordering relation

References

Alan Agresti. *Statistical Methods for the Social Sciences*. Pearson, 5th edition, 2018.

- Daniel J. Benjamin, James O. Berger, Magnus Johannesson, Brian A. Nosek, E.-J. Wagenmakers, Richard Berk, Kenneth A. Bollen, Björn Brembs, Lawrence Brown, Colin Camerer, David Cesarini, Christopher D. Chambers, Merlise Clyde, Thomas D. Cook, Paul De Boeck, Zoltan Dienes, Anna Dreber, Kenny Easwaran, Charles Efferson, Ernst Fehr, Fiona Fidler, Andy P. Field, Malcolm Forster, Edward I. George, Richard Gonzalez, Steven Goodman, Edwin Green, Donald P. Green, Anthony G. Greenwald, Jarrod D. Hadfield, Larry V. Hedges, Leonhard Held, Teck Hua Ho, Herbert Hoijtink, Daniel J. Hruschka, Kosuke Imai, Guido Imbens, John P. A. Ioannidis, Minjeong Jeon, James Holland Jones, Michael Kirchler, David Laibson, John List, Roderick Little, Arthur Lupia, Edouard Machery, Scott E. Maxwell, Michael McCarthy, Don A. Moore, Stephen L. Morgan, Marcus Munafó, Shinichi Nakagawa, Brendan Nyhan, Timothy H. Parker, Luis Pericchi, Marco Perugini, Jeff Rouder, Judith Rousseau, Victoria Savalei, Felix D. Schönbrodt, Thomas Sellke, Betsy Sinclair, Dustin Tingley, Trisha Van Zandt, Simine Vazire, Duncan J. Watts, Christopher Winship, Robert L. Wolpert, Yu Xie, Cristobal Young, Jonathan Zinman, and Valen E. Johnson. Redefine statistical significance. *Nature Human Behaviour*, 2:6–10, 2018. doi: 10.1038/s41562-017-0189-z.
- R. C. Carrasco and J. Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO (Theoretical Informatics and Applications)*, 33(1): 1–20, 1999.
- Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994, Proceedings*, pages 139–152, 1994.
- Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010.
- Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.
- Dakotah Lambert. Relativized adjacency. *Journal of Logic Language and Information*, 32: 707–731, 2023. doi: <https://doi.org/10.1007/s10849-023-09398-x>.
- Dakotah Lambert. System description: A theorem-prover for subregular systems: The language toolkit and its interpreter, plebby. In Jeremy Gibbons and Dale Miller, editors, *Functional and Logic Programming*, pages 311–328, Singapore, 2024. Springer Nature Singapore. ISBN 978-981-97-2300-3.
- Dakotah Lambert, Jonathan Rawski, and Jeffrey Heinz. Typology emerges from simplicity in representations and learning. *Journal of Language Modelling*, 9(1):151–194, 2021.
- Kevin J Lang, Barak A Pearlmutter, and Rodney A Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *International Colloquium on Grammatical Inference*, pages 1–12. Springer, 1998.
- Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.

- José Oncina, Pedro Garcia, et al. Inferring regular languages in polynomial update time. *Pattern recognition and image analysis*, 1(49-61):10–1142, 1992.
- James Rogers and Dakotah Lambert. Some classes of sets of structures definable without quantifiers. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 63–77, Toronto, Canada, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5706. URL <https://www.aclweb.org/anthology/W19-5706>.
- James Rogers and Geoffrey Pullum. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20:329–342, 2011.
- James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. Cognitive and sub-regular complexity. In Glyn Morrill and Mark-Jan Nederhof, editors, *Formal Grammar*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer, 2013.
- Adil Soubki and Jeffrey Heinz. Benchmarking state-merging algorithms for learning regular languages. In François Coste, Faissal Ouardi, and Guillaume Rabusseau, editors, *Proceedings of 16th edition of the International Conference on Grammatical Inference*, volume 217 of *Proceedings of Machine Learning Research*, pages 181–198. PMLR, 10–13 Jul 2023. URL <https://proceedings.mlr.press/v217/soubki23a.html>.
- Katarzyna Stapor. Evaluating and comparing classifiers: Review, some recommendations and limitations. In *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017 10*, pages 12–21. Springer, 2018. doi: 10.1007/978-3-319-59162-9_2.
- Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and Systems Sciences*, 25:370–376, 1982.
- Wolfgang Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, chapter 7. Springer, 1997.
- Sam van der Poel, Dakotah Lambert, Kalina Kostyszyn, Tiantian Gao, Rahul Verma, Derek Andersen, Joanne Chau, Emily Peterson, Cody St. Clair, Paul Fodor, Chihiro Shibata, and Jeffrey Heinz. Mlregtest: A benchmark for the machine learning of regular languages. *Journal of Machine Learning Research*, 25(283):1–45, 2024. URL <http://jmlr.org/papers/v25/23-0518.html>.
- Sicco Verwer and Christian Hammerschmidt. Flexfringe: Modeling software behavior by learning probabilistic automata. *Logical Methods in Computer Science*, 21, 2025.
- Sicco Verwer and Christian A. Hammerschmidt. FlexFringe: A passive automaton learning package. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 638–642, 2017. doi: 10.1109/ICSME.2017.58.