Computational detection of syllable boundaries in undeciphered languages using segment surprisal

Supplementary material

5 Computational models

5.1 Feedforward neural network model

The feedforward neural network is the simplest commonly used neural network architecture, and in many ways the most representative of our initial hypothesis; like the Bayesian model, it operates on individual ngrams, and so can only draw information from local patterns in segment surprisal. By contrast, both the CNN and LSTM models can draw information from across the length of a word, which can improve their predictions in some cases but also makes their results harder to interpret.

As stated, the FNN model operates on individual ngrams, and for each ngram it predicts whether the middle segment is a syllable break or not.[1] It does this by passing the ngram surprisal values through multiple layers of linear transformation and nonlinear activation. Equation 1 shows this process for the first layer,

$$\vec{y} = GELU(\mathbf{W}\vec{x} + \vec{b}) \tag{1}$$

where $\vec{x} = (x_1, x_2, ..., x_n)$ is the surprisal values of the ngram, $\mathbf{W}$ is an $m \times n$ matrix of trainable parameters, $\vec{b} = (b_1, b_2, ..., b_m)$ is a vector of trainable parameters, and GELU is the Gaussian Error Linear Unit (Hendrycks and Gimpel, 2016). In subsequent layers, $\vec{x}$ is replaced with the output $\vec{y}$ from the previous layer, and at the final layer, the model uses sigmoid activation to output a single value $p \in [0, 1]$ to indicate the probability of the middle segment of the ngram being a syllable break. When part of the ngram

[1] We use exclusively odd-numbered ngrams for our FNN and CNN models.

falls outside of the word boundaries (e.g., for the first or last segment of a word), the missing segments' surprisal values are filled in with 0.

Though we call this model an FNN, we implement it using convolutional layers, where the first layer has filters of length n and all subsequent layers have filters of length 1. This is mathematically equivalent to passing each ngram of a word through the FNN described above, and is different from how we implement our CNN model, where the filter length remains fixed at length n for all layers, resulting in an expanding receptive field as described below.

5.2 Convolutional neural network model

The convolutional neural network (CNN) model is a type of neural network architecture that operates by calculating the cross-correlation between its input and a series of filters and then passing this cross-correlation through a non-linear activation function. This outputs a kind of "match" between the filter and the input at each position. These outputs can then be fed as inputs to a new set of filters and non-linear activations for multiple iterations.

$$\vec{y} = GELU(\vec{w} \star \vec{x} + b) \tag{2}$$

Equation 2 shows the output of a single filter $\vec{w}$ over an input $\vec{x}$, where $\vec{x} = (x_1, x_2, ..., x_s)$ is the surprisal values of a word of length s, $\vec{w} = (w_1, w_2, ..., w_n)$ is a vector of trainable parameters of length n, and b is a trainable scalar added to the output at each position; $\star$ is the cross-correlation function and $GELU$ is the Gaussian Error Linear Unit used as a non-linearity (Hendrycks and Gimpel, 2016). Multiple filters are passed over the input at each layer, and their outputs are stacked together.

Another way to view the cross-correlation function is as calculating the dot product of each filter and every ngram of length n in the word. Because the output of one layer is passed as the input to the next, the filters in the next layer are drawing information not just from the outputs of the original ngram, but the outputs of the $n-1$ ngrams to each side of it. This increases the model's receptive field, or the number of segments in the input that can affect the output at a given position, as illustrated in Figure 1. Finally, the last layer of the CNN passes a single filter with length $n = 1$ and Sigmoid activation over the next-to-last layer's output, resulting in a final output $\vec{p} = (p_1, p_2, ..., p_s)$, where each element of $\vec{p} \in [0, 1]$ is interpreted as the probability of the segment in that position

being the start of a new syllable.

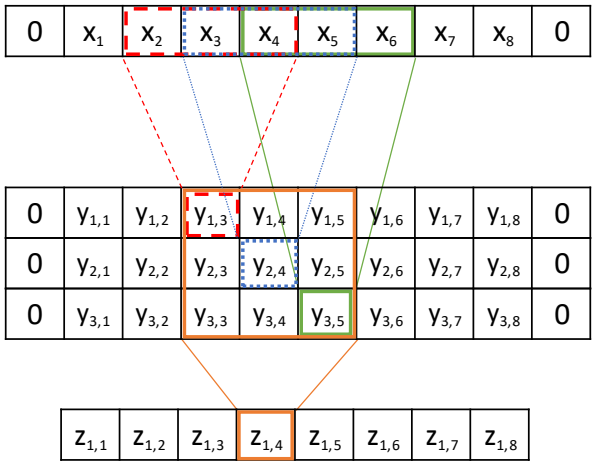

Figure 1: Visualization of two CNN layers. The outputs of the three filters on the input $\vec{x}$ (red, dashed; blue, dotted; green, solid) are then used as inputs to the filter on the second layer $\mathbf{Y}$ (orange, solid). This filter combines multiple outputs from the first layer, as a result combining information from neighboring first-layer ngrams, increasing the model's receptive field.

Importantly, filter length n and number of layers l together determine the model's receptive field. With our particular model, where $n = 7$ and $l = 10$ (see Section 5.4), the receptive field is 67, meaning at any position information from the surrounding 66 segments can filter in and affect the output. Practically speaking, since the longest word in our test dataset is 26 letters, this means that the model takes in information from across the entire word when determining whether a given segment is the start of a new syllable, though it cannot directly "see" the surprisals at each of these positions.

5.3 LSTM model

The long short-term memory (LSTM) model is a subtype of recurrent neural net (RNN) models, which step through an input sequentially while updating an internal "hidden state" that encodes all of the steps seen thus far. They are popular with natural language processing tasks because of the sequential nature of most language data. A vanilla RNN inputting the surprisal values of the word *language*, for example, would start with an initial hidden state $\vec{h}_0$ and the surprisal of the first segment x_1 (i.e. the surprisal of the segment *l* in *language*). It would then update its hidden state with the following equation

$$\vec{h}_t = W_x x_t + \vec{b}_x + W_h \vec{h}_{t-1} + \vec{b}_h$$

where x_t is the input at position t (in this example, $t = 1$), $\vec{h}_{t-1}$ is the hidden state at the previous position (in this case, the initial state $\vec{h}_0$), and W_x, $\vec{b}_x$, W_h, $\vec{b}_h$ are all learned parameters. The model would then move to the second position, x_2 (i.e. the surprisal value of *a* in the second position of *language*), and repeat the same process, now with the updated hidden state $\vec{h}_1$ calculated in the previous step. Thanks to this recurrent process of inputting the hidden state from the previous step $\vec{h}_{t-1}$ along with the current input x_t when calculating the new hidden state h_t, by the time the model arrives at the final segment x_8 (i.e. the surprisal value of *e* in the final position of *language*), the accompanying hidden state $\vec{h}_7$ carries with it information from all of the previous segments in the word.

However, this process only carries information in one direction, i.e. from the beginning of the word to the end. As a result, segments earlier in the word do not benefit from as much information as segments later in the word, and no segment is able to incorporate information from the segments after it. To address this, we use a bidirectional model, which is two RNNs running over the same input, only one starts at the end of the word and works backwards. In this way, the first segment of the word incorporates information from all segments following it in the same way that the final segment of the word incorporates information from all segments preceding it, and any segment in the middle of the word incorporates information from both preceding (from the forward RNN) and following (from the backward RNN) segments.

The LSTM model expands the vanilla RNN model to incorporate memory cells, which store information across multiple steps, and gates to modify the content of those memory cells (Sak, Senior, and Beaufays, 2014). The full set of equations governing the hidden state, memory cell, and gates of the LSTM network are shown below

$$
\begin{aligned}
\vec{i}_t &= \sigma(W_{xi}x_t + \vec{b}_{xi} + W_{hi}\vec{h}_{t-1} + \vec{b}_{hi}) \\
\vec{f}_t &= \sigma(W_{xf}x_t + \vec{b}_{xf} + W_{hf}\vec{h}_{t-1} + \vec{b}_{hf}) \\
\vec{g}_t &= tanh(W_{xg}x_t + \vec{b}_{xg} + W_{hg}\vec{h}_{t-1} + \vec{b}_{hg}) \\
\vec{o}_t &= \sigma(W_{xo}x_t + \vec{b}_{xo} + W_{ho}\vec{h}_{t-1} + \vec{b}_{ho}) \\
\vec{c}_t &= \vec{f}_t \odot \vec{c}_{t-1} + \vec{i}_t \odot \vec{g}_t \\
\vec{h}_t &= \vec{o}_t \odot tanh(\vec{c}_t)
\end{aligned}
$$

where x_t and $\vec{h}_t$ are still the input and hidden state at step t, $\vec{c}_t$ is the memory cell at step t, and $\vec{i}_t$, $\vec{f}_t$, $\vec{g}_t$, and $\vec{o}_t$ are the input, forget, cell, and output gates respectively. $\odot$ represents the Hadamard product, or piecewise multiplication.

The LSTM model consists of multiple of these bidirectional LSTM layers, with each layer taking as input the output of the previous layer. To predict the probability of a syllable break at any given position t, the final layer's hidden state at that position, $\vec{h}_t$, is passed through a linear transformation $y = \vec{w} \cdot \vec{h}_t + b$, where $\vec{w}$ is a vector of learned parameters with the same dimensionality as $\vec{h}_t$ and b is a learned scalar. This linear output y is then passed through the sigmoid function to produce an output between zero and one, which is interpreted as the probability of a syllable break at that position.

Unlike the FNN or CNN models, the LSTM model does not incorporate any concept of ngrams, but instead operates over surprisal values from the entire word. For this reason, it is the largest and most complex of the models included in these experiments.

5.4 Neural network model hyperparameters

Before comparing different model architectures according to their accuracy, we first trained several versions of the same model architecture with different hyperparameters on a train set 502,994 words across all languages. We evaluated these on a held-out validation set of 62,874 words, also sampled from all languages, to gauge what combination of hyperparameters performed best for each model.

The hyperparameters evaluated for each architecture were the ngram length n (FNN and CNN), dimensionality m, and number of layers l. The exact meanings of these hyperparameters differ slightly between architectures, but they are comparable in effect.

Ngram length n refers to the length of inputs fed into the FNN models and to the length of filters in the CNN models; this determines the amount of information each model sees from its input. Dimensionality m indicates the number of neurons in each layer of the FNN, number of filters applied to each layer in the CNN, and dimensionality of the hidden state $\vec{h}$ in the LSTM; this determines the complexity of the model's internal representation of its input. Finally, l is the number of layers per model, not counting an additional input and output layer in the FNN and CNN and an additional output layer in the LSTM.

To see how each hyperparameter affects model performance, we ran a comprehensive grid search over $n = (3,5,7)$, $m = (5,10,15)$, and $l = (5,10,15)$. We also searched outside of this space for specific architectures in preliminary tests, e.g. increasing the ngram length n in FNNs and CNNs to gauge how this could improve model performance before hitting diminishing or no returns. For purposes of comparison, all of our final models are selected from within the hyperparameter space outlined above.

Within this space, we found some general trends, e.g. both FNN and CNN improve as the length of ngrams increases (though this is more pronounced for FNNs), and all three architectures improve as the dimensionality of their internal representations increases. Both CNN and LSTM improve initially with an increase in number of layers from five to ten, but then either stall or perform worse at fifteen layers, while the FNN shows the same performance across all numbers of layers. Table 1 shows the best combinations of hyperparameters for each architecture.

FNN:	n=7	m=15	l=5
CNN:	n=7	m=15	l=10
LSTM:		m=15	l=10

Table 1: Hyperparameters of the FNN, CNN, and LSTM models selected on the validation set.

6 Results

These are the expanded results where we go over experiments not included in our original paper. In Section 6.1, we describe our three training regimes and compare the segment-level metrics of each model under each training regime. In Sections 6.2 and 6.3, we look at segment-level and syllable-level metrics of all three models by position. In Section 6.4, we tease apart the effects of text length and orthography on the CNN

model's predictions specifically. And finally in Section 6.5 we revisit our test case on the UDHR as well as the phonetically transcribed test set with all three models.

6.1 Training regimes

We trained and evaluated each model using three different training regimes. First is the *All Language* training regime, where the model is trained on words from all twelve languages and then evaluated on a held-out test set containing words from those same twelve languages. Second is the *Leave One Out* training regime, where the model is trained on words from eleven languages and then evaluated on a held-out test set consisting only of words from the twelfth, left-out language. Third is the *Language Family* training regime, which is the same as *Leave One Out*, except the model is trained and evaluated only on languages within the same language family. Note that none of the training regimes involve training and evaluating a model on a single language, as is the case for most of the existing literature. Although this would likely show the highest performance for most models, allowing them to overfit on a single language, it is not relevant to our intended application of these models to undeciphered languages.

6.1.1 All Language training regime

For this training regime, we train and validate models on a collection of words taken from all twelve languages and then evaluate them on a held-out test set of words from those same languages. In this way, each model can learn patterns across all languages in the dataset. Although this is not directly comparable to deciphering an undeciphered language, since the model has seen all languages at least once in its training, it gives us a kind of upper bound on how well the models perform cross-linguistically. Comparing the results in Figure 2, we can see that all three models perform similarly on precision, correctly classifying close to half of the segments they predict to be syllable breaks, but CNN and LSTM have significantly higher recall, meaning they miss fewer actual syllable breaks in the words. This suggests that the CNN and LSTM models make many more predictions than the FNN model, while still maintaining roughly the same precision, which we will see illustrated in Figure 8 below.

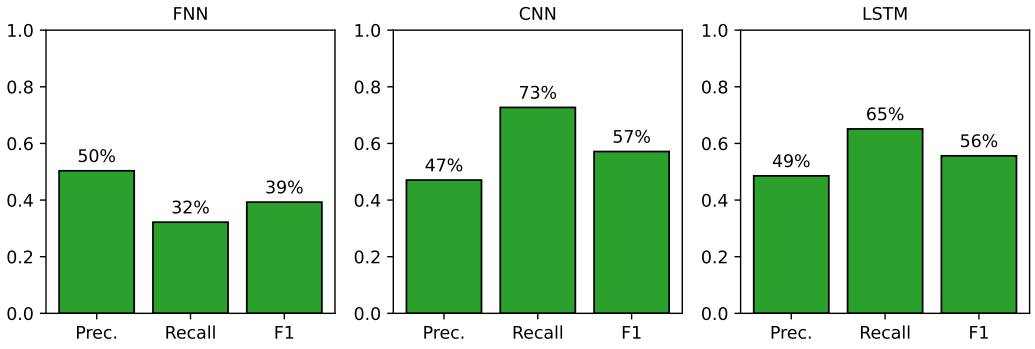

Figure 2: Metrics on the *All Language* training regime calculated across languages and positions.

6.1.2 Leave One Out training regime

For the *Leave One Out* training regime, we train models on all but one of the languages in the dataset, then evaluate it on the left-out language. This is to simulate the practice of training a model on a group of known languages but then running inference on an unknown language for decipherment. Comparing these results to the *All Language* training regime can give us a better idea of the performance we might expect to see in a real-world decipherment scenario.

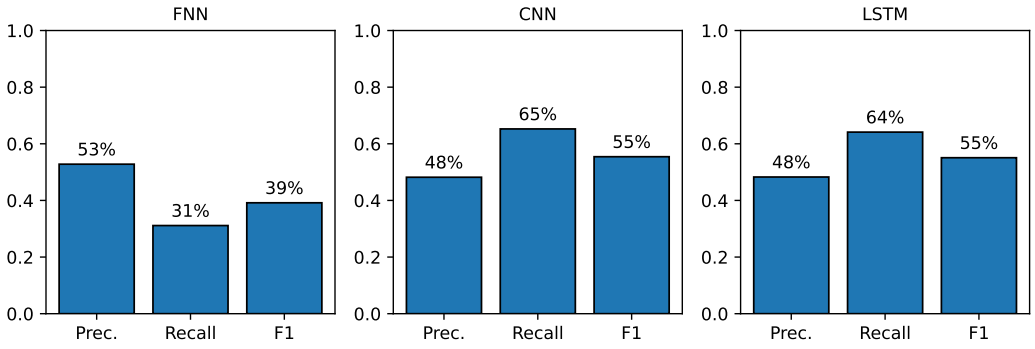

Figure 3: Metrics on the *Leave One Out* training regime calculated across languages and positions.

Figure 3 shows the precision, recall, and F1 calculated across all languages, where each language is predicted by a model trained on all languages except the predicted one. We can see that LSTM and CNN performance decreases significantly in this training regime, falling slightly below FNN in precision but remaining higher than it in recall and F1. FNN performance remains fairly stable between the *All Language* and *Leave*

One Out training regimes, though it drops slightly. Appendix **??**) shows these models' performance split by language.

6.1.3 Language Family training regime

The last training regime is the *Language Family* regime. This regime employs the same methods as *Leave One Out*, but instead of training on all other languages within the dataset, it only trains on other languages in the same family as the left-out language. For our dataset, this means we train on the West Germanic languages (English, Dutch, German), North Germanic languages (Swedish, Norwegian), and Romance languages (French, Italian, Spanish). It also means that our training sets are significantly smaller, as each training set will consist of only one or two other languages.[2] This simulates the case in which the language family of an undeciphered language is known or assumed.

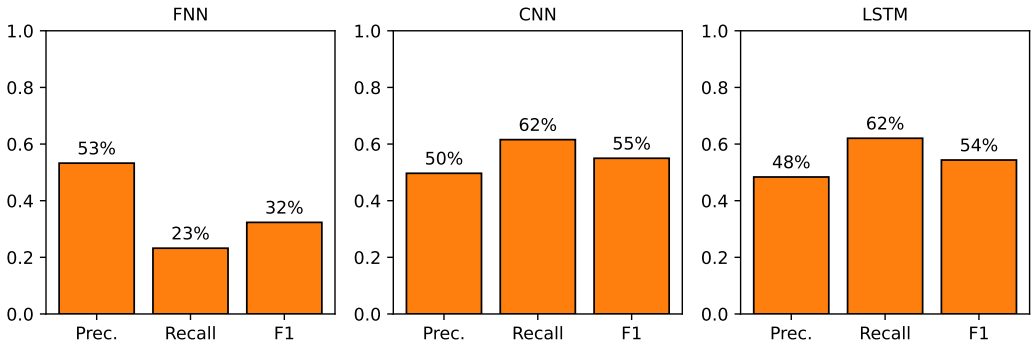

Figure 4: Metrics on the *Language Family* training regime calculated across languages and positions.

The performance of the *Language Family* regime indicates (Figure 4) that it provides a better alternative to *Leave One Out* training when the family of an undeciphered language is known, which likely stems from intra-family effects or phonotactic similarities between languages. However, this is also influenced by the fact that the *Language Family* regime omits Czech, Greek, Korean, and Turkish, which don't have related languages in the dataset, from training and evaluation, which may skew the results.

[2]For *Leave One Out* training, we also removed one language from the training set each time, but then supplemented the training set with extra words taken from the validation (not test) set.

6.1.4 Analysis of training regime results

Comparing each model's performance across the three training regimes allows us to see how each model responds to losing a specific language's information from its training data and to speculate why the different models react differently to this loss of information.

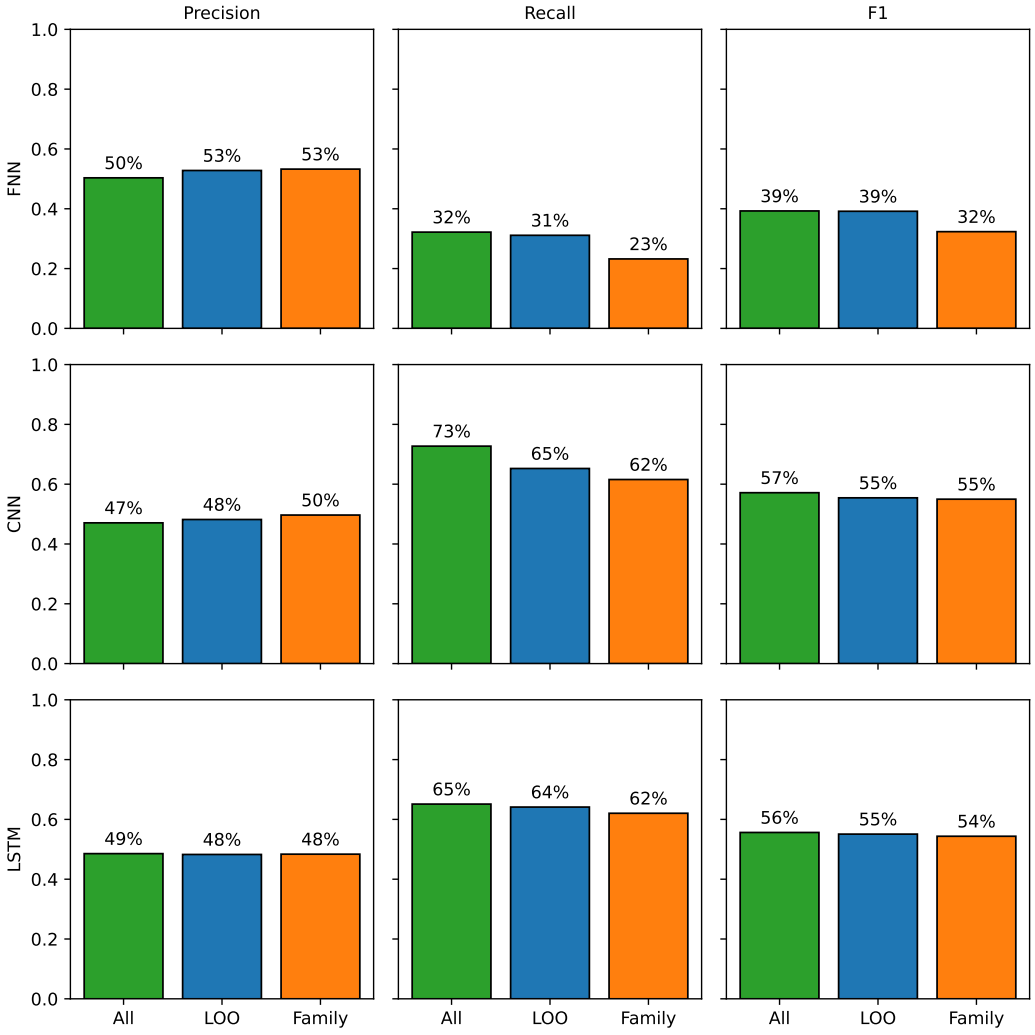

Figure 5: Metrics compared across all three training regimes for each model.

It is apparent from Figure 5 that, while FNN performs worse overall than LSTM and CNN on all training regimes (with the exception of precision), its performance across all three training regimes is the most consistent, i.e. it is not as much affected by the loss of a single language's information from its training data. By contrast, LSTM and CNN

perform better than FNN (especially on the *All Language* training regime), but their performance declines when the predicted languages are not present in their training data. This suggests that the learned patterns, while more complex and predictive than those of the FNN, are also more language-specific and therefore do not generalize as well to unknown languages. The FNN models, meanwhile, seem to learn less complex patterns that are less predictive but generalize better to unknown languages, so that when a language is removed from the dataset, it does not affect their performance on that language as severely.

6.2 Segment-level accuracy

As noted in the main paper surprisal decreases across the length of a word. Because of this, we might wonder whether our models, which predict syllable breaks based on patterns of surprisal, become less effective at later positions in a word, especially where surprisal drops to zero. Figure 6 shows the precision, recall, and F1 of each model at each position in a word[3]. For the FNN, we see the number of syllable break predictions drops sharply with position, and with it the recall of the model also drops (since it is missing many true syllable breaks that occur at later positions in a word). This is not surprising, as the FNN only has an ngram window (in this case, a 7-gram) from which to predict a syllable break. If all the segments in that neighborhood have a surprisal of zero, as may be the case near the end of a word, it has nothing to base its prediction on and therefore does not predict any syllable breaks. The CNN by contrast, though it initially works with a window of the same size (7-gram), through its hierarchical structure is able to draw information from neighboring ngrams, and we see that the number of predicted syllable breaks actually exceeds the number of true syllable breaks at almost all positions past two. Recall remains high at these later positions, though precision suffers somewhat, perhaps due to the excessive number of predictions. Finally, the LSTM, which for every segment is able to see surprisal at all positions in the word, predicts the closest to the actual number of syllable breaks at each position and maintains good precision and recall at all positions.

[3]Figure 6 shows metrics for the *Leave One Out* training regime specifically; results in the other training regimes are very similar

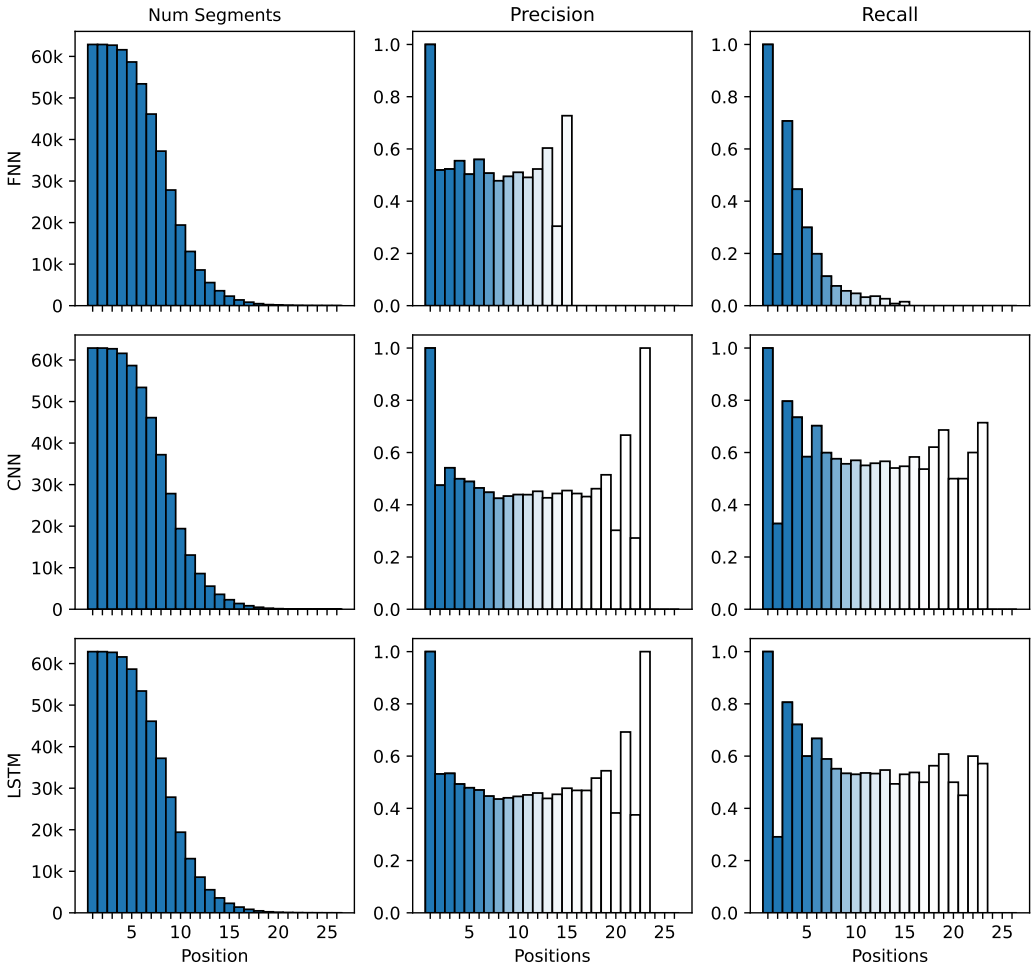

Figure 6: Number of segments and segment-level precision and recall on the *Leave One Out* training regime, calculated per position. Opacity in the precision and recall graphs is proportional to the number of segments at each position, so positions with fewer segments are less opaque.

Further, Figure 7 contrasts the percentage of predicted and true syllable breaks by position and entropy across all three models.

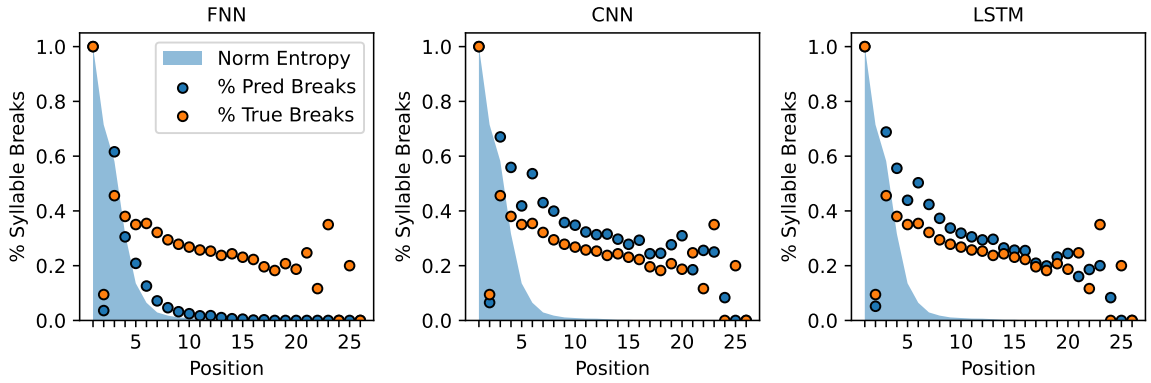

Figure 7: Percentage of segments at each position in a word that are syllable breaks or predicted to be syllable breaks by the models. In the background is entropy (average surprisal), normalized here between 0 and 1.

The better performance of CNN and LSTM at later positions in a word, especially in terms of recall, helps explain their overall better performance in Figure 5. However it's important to note that the majority of segments (58%) are in the first five positions, as are the majority of syllable breaks even discarding the first position (54%), so the models' performance on the beginnings of words are what carry the most weight for overall segment-level evaluation.

6.3 Syllable-level accuracy

To further analyze the models' prediction accuracy, we need to investigate the accuracy on a syllable level. This is important since, in a real-world decipherment scenario, we expect the model to return syllables that are indeed syllables in the language in question. For this reason, we will mostly focus on *precision*, i.e., what proportion of syllables the model identifies are in fact syllables in the language, though we will also look at recall to see how many of the actual syllables of the language are captured by the model.

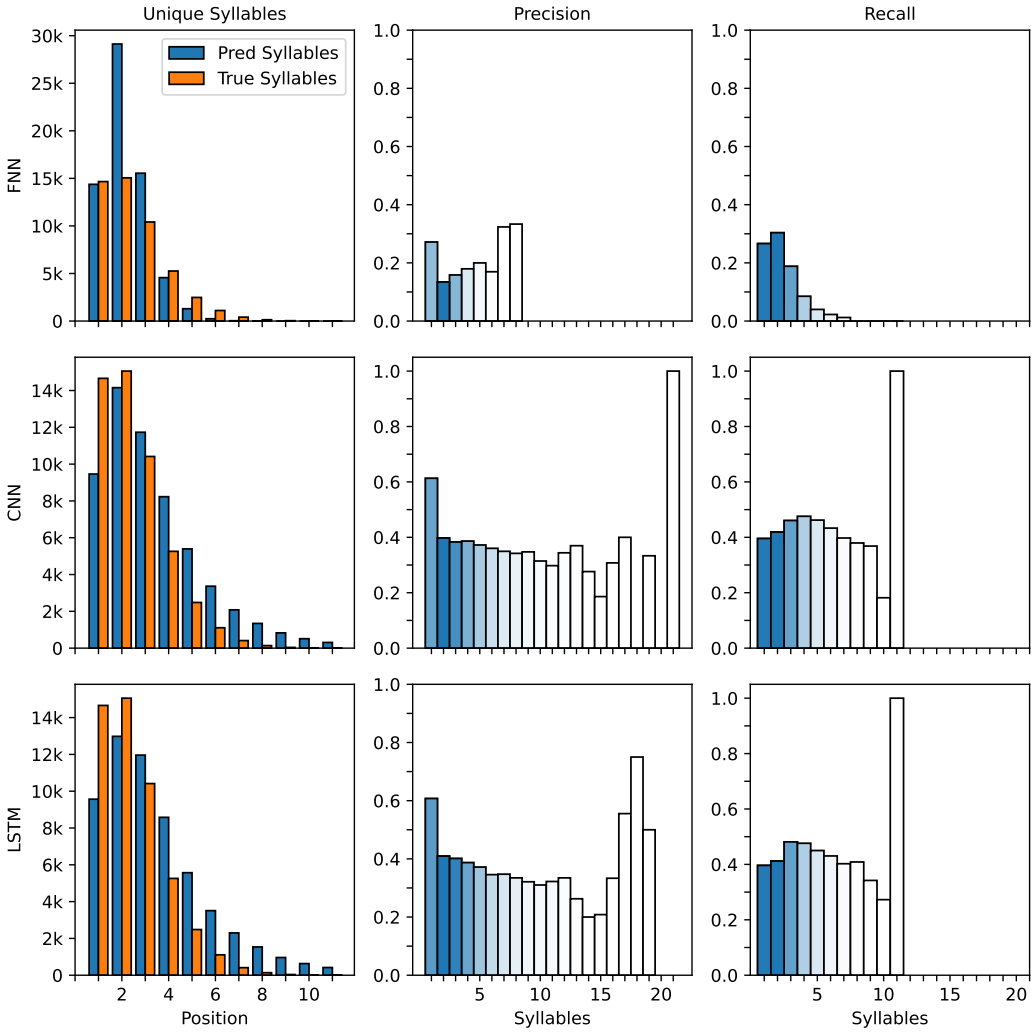

Figure 8: Unique syllables from the *Leave One Out* training regime, with metrics calculated per position. Position 1 means the first syllable in a word, position 2 the second, etc.

For this, we used models trained on the *Leave One Out* training regime and look at the number of *unique* syllables in both the predictions and ground truths. Taking note of the positional analysis for segment-level accuracy, we first plot the precision and recall of predicted syllables based on their position in the word (Figure 8), and find that for all three models the first syllable of a word is predicted with much higher precision (60%) than later syllables (generally $<$ 40%). For this reason, we restricted further evaluation to only word-initial syllables. We extracted the predicted syllables for every model that occur more frequently than 10 times in the first position to remove outliers and one-off predictions. As a next step, we calculated the ratio between true positive

syllables and the total number of predictions (precision score) to gauge how the models would perform in a real-world scenario. If the syllables commonly identified by the model are mostly true syllables, we can be confident that, in a test case with unknown syllabification, the majority of the model-identified syllables will be actual syllables. To establish a random baseline, we added predictions of two simple baseline models. The first *Random* is a random prediction model which guesses syllable breaks at random in the test data. The second is the *PeakIdentifier* baseline which guesses syllable breaks by whether the segment in question is a syllable peak or not. The latter model is an entirely rule-based implementation of the theory discussed in Section 3 of the main paper. Table 2 shows the average precision and recall scores and standard deviation across languages, stratified by model and syllable length.

			Syllable length			
Metric	Model	Total	1	2	3	4
precision	CNN	0.81 (0.11)	0.616 (0.2)	0.84 (0.11)	0.813 (0.17)	0.56 (0.46)
	FNN	**0.82** (0.11)	**0.672** (0.24)	0.841 (0.11)	0.851 (0.15)	0.536 (0.41)
	LSTM	0.817 (0.11)	0.517 (0.23)	**0.851** (0.11)	**0.877** (0.15)	**0.614** (0.46)
	Random	0.53 (0.13)	0.315 (0.1)	0.79 (0.13)	0.733 (0.43)	N/A
	PeakIdentifier	0.701 (0.13)	0.209 (0.1)	0.825 (0.13)	0.79 (0.16)	0.47 (0.36)
recall	CNN	0.051 (0.03)	0.414 (0.23)	0.249 (0.1)	0.015 (0.01)	0.001 (.0)
	FNN	**0.054** (0.03)	0.268 (0.25)	**0.257** (0.11)	**0.019** (0.02)	0.001 (.0)
	LSTM	0.051 (0.03)	0.396 (0.24)	0.241 (0.1)	0.018 (0.01)	0.001 (.0)
	Random	0.018 (0.01)	**0.643** (0.14)	0.08 (0.05)	0 (.0)	0 (.0)
	PeakIdentifier	0.036 (0.03)	0.241 (0.18)	0.141 (0.08)	0.017 (0.02)	**0.002** (.0)

Table 2: Average precision and recall scores across languages by model and syllable length. Values in parentheses indicate standard deviation over all language sets. Figures in boldface indicate the best model per column.

We can see that only syllables of lengths 2 and 3 are predicted with good precision. Syllables of length one are being overpredicted and perform at or below random, meaning that the model identifies more syllables than there are true one-segment syllables in the datasets. For recall, all models drop off significantly at syllables of length 3 or more, meaning they only predict a small number of syllables compared to the total number of syllables in the language. A good precision-recall combination can only be found in syllables of length 2. In overall recall, all NN models outperform the

rule-based model. However, this difference is strongest in the recall values of syllables of length 2 and 3. In precision, all models outperform the PeakIdentifier and the random baseline. We can, moreover, conclude that the FNN is the best overall model with the LSTM being better for syllables of sizes longer than one.

6.4 Effects of text length and orthography

Figure 9 shows the entropy distribution by position of the train set, test set, and the UDHR test case. This indicates that the smaller-sized datasets have shorter and lower entropy tails for later positions.

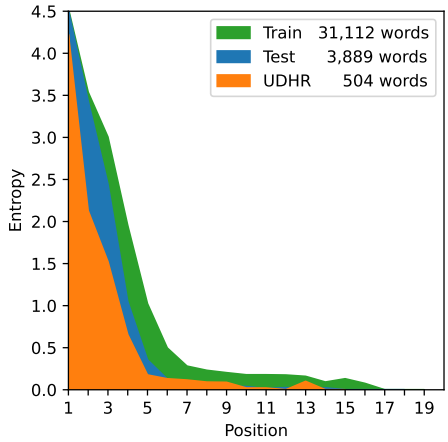

Figure 9: Entropy over the English train set (31,112 words), test set (3,889 words), and UDHR (504 words).

To visualize the relationship between sample size of the test set and precision/recall, Figure 10 shows a comprehensive overview.

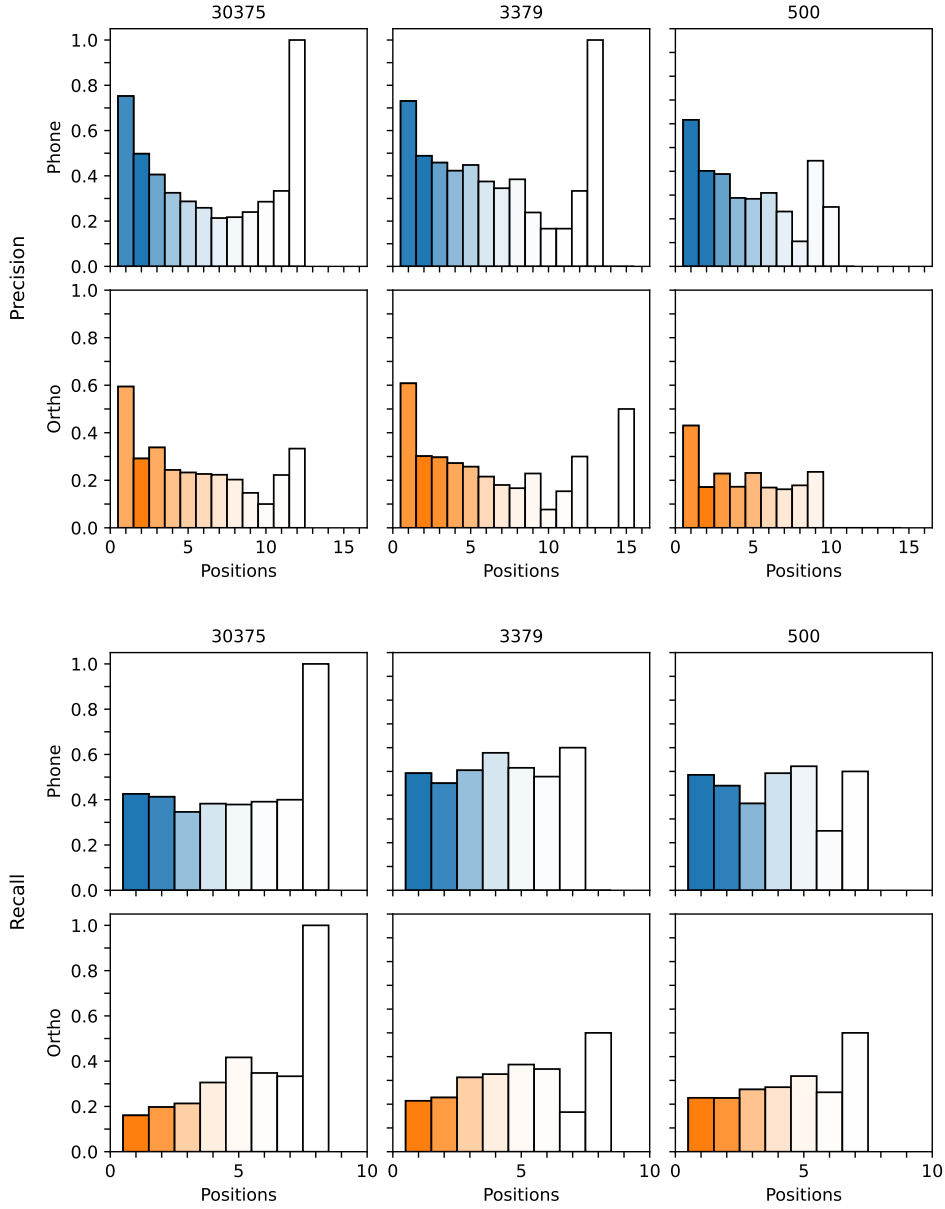

Figure 10: Precision and Recall on a subset of the test set at different sizes, with phonetic transcriptions and orthography.

The effect of smaller sample sizes is similar in precision and recall across both orthographic and phonetic transcriptions. Mainly, smaller sample sizes reduce precision and increase recall by a small margin.

6.5 Predicted syllables on the phonological data of English

dʒɪ	(br)	(fl)	hʌ	kaʊ	mæ	pɪ	saɪ	tɛ
æn	dɛ	fɒ	ɪm	(kl)	məʊ	(pl)	sɪ	(tr)
əb	di	(fr)	ɪmp	kɒ	mɛ	(pr)	(sk)	ʌn
əd	dɪ	gæ	ɪn	kɒn	mɪ	prɪ	(sl)	wɔː
ək	dɪs	(gr)	ju	(kr)	mɪs	ræ	(sm)	waɪ
(əp)	(dr)	hæ	kæ	læ	mʌ	rəʊ	(sp)	wɛ
aʊt	fæ	hɑː	kə	laɪ	pæ	rɛ	(st)	wɪ
bæ	fə	haɪ	kən	ləʊ	pə	ri	(str)	
bɪ	fɔː	hɛ	kɔː	lɛ	pɜː	rɪ	sʌ	
(bl)	fɪ	hɒ	kəʊ	lɪ	pɛ	sæ	tæ	

Table 3: Most common ($>$ 90 percentile of all syllables) two-segment and three-segment syllables predicted by the FNN model for the English test dataset. Syllables in parentheses are false positives (i.e., not contained in the test dataset), while asterisks indicate the two forms ər which are coded differently in the dataset based on whether there can be a linking-r in the British phonotactic system. Here, 69 are true syllables, 18 ($\approx$ 20%) are false.

dʒɪ	dɪ	haɪ	(kr)	pæ	sɪ
æn	dɪs	hɛ	(kw)	pə	(sk)
əb	(dr)	hɒ	lɛ	pɪ	(sp)
ək	fæ	ɪk	lɪ	(pl)	(st)
aʊt	fə	ɪm	mæ	(pr)	(str)
bæ	fɔː	ɪn	mə	ræ	tæ
bɪ	fɪ	kæ	mɛ	rəʊ	tɛ
(bl)	(fl)	kə	mɪ	rɛ	(tr)
(br)	(fr)	kən	mɪs	ri	ʌn
dɛ	gæ	(kl)	mɒ	rɪ	wɛ
di	(gr)	kɒ	mʌ	sɛ	wɪ

Table 4: Most common ($>$ 90 percentile of all syllables) two-segment and three-segment syllables predicted by the CNN model for the English test dataset. Syllables in parentheses are false positives (i.e., not contained in the test dataset), while asterisks indicate the two forms ər which are coded differently in the dataset based on whether there can be a linking-r in the British phonotactic system. Here, 50 are true syllables, 16 ($\approx$ 15%) are false.

dʒɪ	di	haɪ	kɒ	mʌ	(pr)	(sm)
æn	dɪ	ɪk	(kr)	pæ	ræ	(sp)
əb	dɪs	ɪm	kʌ	pə	rəʊ	(st)
əd	(dr)	ɪn	(kw)	pɑː	rɛ	sʌ
aʊt	fɔː	kæ	mæ	paɪ	ri	tæ
bæ	fɪ	kə	mə	pɜː	sæ	tɛ
bɪ	(fl)	kən	mɛ	pɛ	saɪ	(tr)
(bl)	(fr)	kɔː	mɪ	pɪ	sɛ	træ
(br)	gæ	kəʊ	mɪs	(pl)	sɪ	ʌn
dɛ	(gr)	(kl)	mɒ	pɒ	(sk)	wɪ

Table 5: Most common (> 90 percentile of all syllables) two-segment and three-segment syllables predicted by the LSTM model for the English test dataset. Syllables in parentheses are false positives (i.e., not contained in the test dataset), while asterisks indicate the two forms ər which are coded differently in the dataset based on whether there can be a linking-r in the British phonotactic system. Here, 54 are true syllables, 16 (≈ 20%) are false.