# CODE-SWITCHED LANGUAGE MODELLING USING A CODE PREDICTIVE LSTM IN UNDER-RESOURCED SOUTH AFRICAN LANGUAGES

*Joshua Jansen van Vüren, Thomas Niesler*

Department of Electrical and Electronic Engineering,
Stellenbosch University, Stellenbosch, South Africa

## ABSTRACT

We present a new LSTM language model architecture for code-switched speech incorporating a neural structure that explicitly models language switches. Experimental evaluation of this code predictive model for four under-resourced South African languages shows consistent improvements in perplexity as well as perplexity specifically over code-switches compared to an LSTM baseline. Substantial reductions in absolute speech recognition word error rates (0.5%-1.2%) as well as errors specifically at code-switches (0.6%-2.3%) are also achieved during n-best rescoring. When used for both data augmentation and n-best rescoring, our code predictive model reduces word error rate by a further 0.8%-2.6% absolute and consistently outperforms a baseline LSTM. The similar and consistent trends observed across all four language pairs allows us to conclude that explicit modelling of language switches by a dedicated language model component is a suitable strategy for code-switched speech recognition.

***Index Terms—*** Code-switching, Bantu languages, n-best rescoring, language model data augmentation, speech recognition, under-resourced languages

## 1. INTRODUCTION

Code-switching is the practice of utilising multiple languages within and between sentences and occurs predominately in spontaneously spoken speech in multilingual communities. Improving code-switched speech recognition and language modelling is an active research area, but has proved to be a challenging task. The phenomenon is speaker dependent and occurs rarely when compared with monolingual segments of speech and text. As a result, corpora of code-switched speech are generally under-resourced.

Various techniques have been proposed to improve speech recognition and language modelling in the presence of code-switching. These include transliteration [1, 2, 3], data-augmentation using linguistic techniques [4, 5] or neural networks, and improved language and acoustic model architectures [6, 7].

Data augmentation by synthetic generation of code-switched text using neural networks is regularly applied. LSTM language models [8, 9, 10, 11], generative adversarial networks [12], variational auto-encoders [13], as well as state of the art transformers [14] and hybrid bidirectional adversarial transformer networks [15] have been successfully employed to synthesize code-switched text. Such text can be used to augment training sets for n-gram and neural language models.

Generation methods premised on linguistic theories such as matrix language frame theory [16] and equivalence constraint theory [17] have also been shown to successfully generate artificial code-switched text [4, 5]. Combinations of both neural and linguistically-based generative techniques provide further gains [18].

Architectural adaptations have been considered to optimise the performance of code-switched language models and speech recognition systems. For example, perplexity and speech recognition for code-switched English-isiZulu was improved by utilising language dependent language and acoustic models in [19]. Language dependent acoustic and language models differentiate between the phones and word tokens of different languages, while no such distinction is made in language independent models. Distinct improvements in English-Mandarin speech recognition at code-switch points have been achieved through two adaptations of recurrent neural networks applied in n-best rescoring [6]. Firstly, the output layer is split to model the output word as well as one of four classes: particles, English, Mandarin, and anything else. The second adaptation splits the input layer in order to encode both the input word and its part-of-speech.

A novel recurrent neural network that utilises two LSTM language models to separately model monolingual segments was presented in [7]. Here the language of the current input word token is used to select the LSTM that models the next word. This dual LSTM language model was shown to reduce test set perplexity. Further perplexity improvements were achieved when the model was used to generate synthetic data for pre-training.

Our work focuses on developing an improved language model architecture specifically for code-switched speech, and that can be applied in both n-best rescoring and data augmentation. In contrast to the research presented in [7], our proposed network contains a specific neural structure that explic-

978-1-6654-7189-3/22/$31.00 

itly models the code of the next word.

The remainder of the paper is organised as follows. Section 2 describes the code-switched dataset utilised in this work. Section 3 motivates and describes the proposed code predictive model. Section 4 details our experimental setup. Section 5 discusses the results of the architectural optimisation and presents results from initial n-best rescoring experiments. Section 6 presents speech recognition experiments in which the model is utilised to synthesize text which is used to augment n-gram language model training sets. Finally, Section 7 concludes.

## 2. DATASET

Four bilingual sub-corpora of manually transcribed code-switched speech constitute the dataset used in this work. The speech was compiled from South African soap opera episodes [20], and each sub-corpus is made up of a Bantu language and English, namely English-isiZulu (EZ), English-isiXhosa (EX), English-Sesotho (ES), and English-Setswana (ET), as shown in Table 1. Previous work utilising the same dataset found that incorporating out-of-domain monolingual corpora into n-gram language models training sets chiefly improves the English word error rate, often at the expense of the error rate for the under-resourced language [11]. Therefore, we do not utilise such corpora in this work.

**Table 1**. The soap opera corpus, showing the total number of word tokens, word types, and code switches in the four bilingual sub-corpora. $CS_{EB}$ indicates the number of switches from English to a Bantu language, while $CS_{BE}$ indicates switches from Bantu to English. The final column indicates audio duration in hours (h) or minutes (m).

| Pair | Partition | Tok | Typ | $CS_{EB}$ | $CS_{BE}$ | Dur |
|---|---|---|---|---|---|---|
| English-isiZulu (EZ) | Train | 52383 | 10396 | 2236 | 2743 | 4.81h |
| | Dev | 1566 | 866 | 175 | 198 | 8.00m |
| | Test | 5656 | 2305 | 688 | 776 | 30.4m |
| English-isiXhosa (EX) | Train | 32539 | 7716 | 776 | 1003 | 2.68h |
| | Dev | 2300 | 1246 | 91 | 113 | 13.7m |
| | Test | 2651 | 1387 | 328 | 363 | 14.3m |
| English-Sesotho (ES) | Train | 35197 | 4339 | 1565 | 1719 | 2.36h |
| | Dev | 3067 | 1050 | 156 | 166 | 12.8m |
| | Test | 4054 | 1193 | 403 | 396 | 15.5m |
| English-Setswana (ET) | Train | 35725 | 3808 | 1885 | 1951 | 2.33h |
| | Dev | 3707 | 1052 | 224 | 251 | 13.8m |
| | Test | 4939 | 1254 | 505 | 526 | 17.8m |

## 3. CODE PREDICTIVE LANGUAGE MODEL

This section describes the structure of our proposed code predictive model. The goal of this network is to explicitly model code-switching by structurally encoding the language switch into the network, rather than relying on this phenomenon to be learnt implicitly from the extremely sparse data.

Previous research has highlighted that a chief challenge in language modelling of code-switched text remains the very high confusion (in terms of perplexity) at language switches, which contrasts with the much lower perplexity within the monolingual segments [11]. It is possible, therefore, that an architectural mechanism that specifically models code-switches may benefit the performance of the language model. In our proposed model, we refer to this additional component as the *code predictor*, and its function is to make a binary selection between two languages.

### 3.1. Language Model Structure

Structurally, our language model is made up of four components: a word embedding layer, a code predictor, and two monolingual language models, as is illustrated in Figure 1.

The figure shows two sets of vectors which characterize the flow of information through the network. These are the language model state vectors ($\boldsymbol{h}_{\text{LM}}^{(i-1)}$,$\boldsymbol{c}_{\text{LM}}^{(i-1)}$) and the code predictor state vectors ($\boldsymbol{h}_{\text{CP}}^{(i-1)}$,$\boldsymbol{c}_{\text{CP}}^{(i-1)}$). For each word in a sequence, the code predictor receives the embedding vector ($\boldsymbol{x}_{\text{CP}}^{(i)}$) of the current word and its previous state vectors, $\boldsymbol{h}_{\text{CP}}^{(i-1)}$ and $\boldsymbol{c}_{\text{CP}}^{(i-1)}$, as inputs. On the basis of these inputs, the code predictor updates its state vectors. The updated hidden state vector $\boldsymbol{h}_{\text{CP}}^{(i)}$ is transformed to a single neuron by a dense layer with a sigmoid activation function. This acts as a binary classifier, indicating the most likely language of the next word as a scalar $l^{(i)}$ in the range $0 \leq l^{(i)} \leq 1$. From this output, a loss can be computed with respect to the ground truth language ($\tilde{v}^{(i)}$).

Similarly, for each successive word, each language model receives the embedding vector $\boldsymbol{x}^{(i)}$ of the current word and the previous state vectors $\boldsymbol{h}_{\text{LM}}^{(i-1)}$ and $\boldsymbol{c}_{\text{LM}}^{(i-1)}$. The updated hidden states ($\boldsymbol{h}_{\text{LM,E}}^{(i)}$ and $\boldsymbol{h}_{\text{LM,B}}^{(i)}$) of the two language models are passed to an associated dense layer which transforms these to logit vectors ($\boldsymbol{o}_{\text{E}}^{(i)}$, $\boldsymbol{o}_{\text{B}}^{(i)}$) with the dimensionality of the combined bilingual vocabulary $d_{\text{vocab}}$. To form the likelihood vector $\boldsymbol{o}^{(i)}$, the vectors $\boldsymbol{o}_{\text{E}}^{(i)}$ and $\boldsymbol{o}_{\text{B}}^{(i)}$ are interpolated according to the probability of the next language as modelled by the output $l^{(i)}$ of the code predictor. From this output, a loss can be computed with respect to the ground truth ($\boldsymbol{v}^{(i)}$), or the argmax can be taken to sample the model prediction. Preliminary experiments showed that individually accumulating and descending gradients associated with the language predictions and language models produced the best performing models.

The language predictor output ($l^{(i)}$) is also used to select which language model state vectors, ($\boldsymbol{h}_{\text{LM,E}}^{(i)}$,$\boldsymbol{c}_{\text{LM,E}}^{(i)}$) or ($\boldsymbol{h}_{\text{LM,B}}^{(i)}$,$\boldsymbol{c}_{\text{LM,B}}^{(i)}$), are used as the state vectors ($\boldsymbol{h}_{\text{LM}}^{(i)}$,$\boldsymbol{c}_{\text{LM}}^{(i)}$) for the next word.

### 3.2. Language Model Components

This network structure enables the code predictor to explicitly select the monolingual LSTM with which to compute the probability of the next word. Therefore, the code predictor deals only with the code selection while the language model

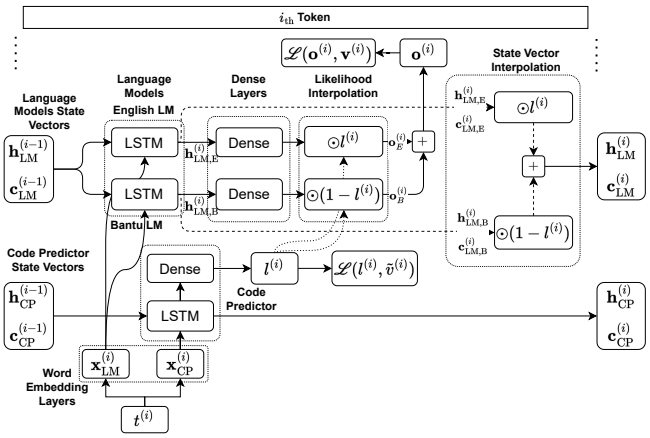


**Fig. 1**. The structure of the proposed code-switched network architecture $\mathbb{B}$ described in Section 3. The tokens $t^{(i)}$ from the input sequence $(t^{(0)} \dots t^{(n-1)})$ are embedded ($\boldsymbol{x}_{\text{LM}}^{(i)}$ and $\boldsymbol{x}_{\text{CP}}^{(i)}$) and presented as input to the three LSTM models. The network uses two pairs of state vectors, denoted $\boldsymbol{h}_{\text{LM}}^{(i)}$,$\boldsymbol{c}_{\text{LM}}^{(i)}$ and $\boldsymbol{h}_{\text{CP}}^{(i)}$,$\boldsymbol{c}_{\text{CP}}^{(i)}$ respectively. The predicted language for token $t^{(i+1)}$ is given by the scalar $l^{(i)}$ ($0 \leq l^{(i)} \leq 1$), and acts as a mask to select among the two logit vectors $\boldsymbol{o}_{\text{E}}^{(i)}$ and $\boldsymbol{o}_{\text{B}}^{(i)}$ which are a transformation of the LSTM language model hidden states $\boldsymbol{h}_{\text{LM,E}}^{(i)}$ and $\boldsymbol{h}_{\text{LM,B}}^{(i)}$ respectively. $\odot$ denotes the element-wise product.

deals only with words in the respective language. In this way the code predictor learns to model the language switches, while the monolingual language models observe tokens from both languages as input but respectively model monolingual word sequences. This aspect of our approach is novel.

#### *3.2.1. Word Embeddings*

The word embedding layer associates each word type in the combined bilingual vocabulary with a point in a continuous vector space of dimension $d_{\text{embed}}$. As also listed in Table 2, we consider both an architecture ($\mathbb{B}$) which has separate embedding layers for the code predictor $\boldsymbol{x}_{\text{CP}}^{(i)}$ and the language models $\boldsymbol{x}_{\text{LM}}^{(i)}$ (shown in Figure 1) and an architecture ($\mathbb{C}$) that shares the embedding layer between the three LSTMs. Sharing the embedding layer between the language models and the code predictor could lead to greater predictive ability due to an effect reminiscent of multitask training.

#### *3.2.2. Code Predictor*

For each word in a sequence, the code predictor receives the state vectors ($\boldsymbol{h}_{\text{CP}}^{(i-1)}$,$\boldsymbol{c}_{\text{CP}}^{(i-1)}$) from the previous word, as well as the embedding vector $\boldsymbol{x}_{\text{CP}}^{(i)}$ of the current word.

We consider an architecture ($\mathbb{C}$) in which the code predictor and the two language models have separate state vectors, and an architecture ($\mathbb{D}$) in which these vectors are shared, therefore making $\boldsymbol{h}_{\text{LM}}^{(i)} = \boldsymbol{h}_{\text{CP}}^{(i)}$ and $\boldsymbol{c}_{\text{LM}}^{(i)} = \boldsymbol{c}_{\text{CP}}^{(i)}$. When the state vectors are shared, they are first passed to and updated by the code predictor, and then passed to the language models.

#### *3.2.3. Language Models*

For each word in a sequence, both language models update their respective state vectors ($\boldsymbol{h}_{\text{LM}}^{(i)}$, $\boldsymbol{c}_{\text{LM}}^{(i)}$) and likelihood vectors ($\boldsymbol{o}_{\text{E}}^{(i)}$, $\boldsymbol{o}_{\text{B}}^{(i)}$). We investigate whether or not a hard interpolation ($l^{(i)}$ is strictly binary and takes on a value of either 0 or 1) is preferable to a soft interpolation for both the token likelihood predictions ($\mathbb{E}$) and the state vectors ($\mathbb{F}$).

Finally, we also consider an architecture $\mathbb{G}$ in which we add a language embedding layer $\boldsymbol{y}^{(i)}$ which associates each language (English and Bantu) with a respective embedding. This language embedding is concatenated with the token embedding as input to the language models and the code predictor.

## 4. EXPERIMENTAL SETUP

Our experiments consider text synthesis for n-gram augmentation and automatic speech recognition by means of 50-best list rescoring.

### 4.1. Baseline Acoustic Model

The acoustic model in our experiments is a CNN-TDNN-F model first pre-trained on all the data in the four sub-corpora (Table 1) and then fine-tuned on the respective bilingual pairs. This acoustic model was found to offer better performance than several other considered alternatives, including a state of the art Wav2Vec2 architecture [21].

We denote the baseline KALDI system using an n-gram language model as $\mathbb{A}_{\phi}$ [22]. The n-gram in question is a trigram with modified Kneser-Ney smoothing trained only on the respective training sets shown in Table 1 using the SRILM toolkit [23]. The vocabulary is closed over all types in each respective bilingual sub-corpus.

### 4.2. Baseline Language Models

Our baseline LSTM (system $\mathbb{A}$) contains a single embedding layer for the input tokens $t^{(i)}$, a single LSTM layer, and a dense layer with the dimensionality of the vocabulary ($d_{\text{vocab}}$) such that the network models the likelihood of the next word.

The baseline LSTM and code predictive models were trained on the bilingual training sets shown in Table 1 until optimal cross-entropy loss was achieved on the development set. Additionally, the following hyperparameters were fixed: 256 embedding dimensions for both word and language embeddings, 256 recurrent dimensions, a batch size of 32, and L2 weight regularisation was utilised. During n-best rescoring, the interpolation weight assigned to the n-gram and the neural language model were optimised using the development set word error rate.

In order to ensure that the baseline is comparable in terms of overall number of parameters. We also considered the effectiveness of deeper and wider LSTM models. When the

**Table 2**. Baseline LSTM ($\mathbb{A}$), default code predictive ($\mathbb{B}$) and alternative architectural configurations ($\mathbb{C}$-$\mathbb{G}$) considered for optimisation.

| Alias | Configuration |
|---|---|
| $\mathbb{A}_\phi$ | Baseline n-gram |
| $\mathbb{A}$ | Baseline LSTM |
| $\mathbb{B}$ | Separate embeddings per LSTM (Default) |
| $\mathbb{C}$ | $\mathbb{B}$ with shared embeddings between LSTMs |
| $\mathbb{D}$ | $\mathbb{C}$ with shared hidden and cell state vectors |
| $\mathbb{E}$ | $\mathbb{D}$ with soft interpolation of language logits |
| $\mathbb{F}$ | $\mathbb{E}$ with soft interpolation of hidden and cell state vectors |
| $\mathbb{G}$ | $\mathbb{E}$ with additional language embedding |

state vectors are shared (system $\mathbb{D}$), the code-predictive language model can be interpreted as a two layer deep LSTM that has a larger recurrent dimensionality than the baseline LSTM (system $\mathbb{A}$). Therefor, we trained a two layer LSTM network, as well as an LSTM affording the same recurrent dimensionality as the three LSTMs constituting the code-predictive network shown in Figure 1. However, neither alternative provided improvements over the baseline LSTM (system $\mathbb{A}$). Therefore, system $\mathbb{A}$ was chosen as the baseline LSTM implementation in this study.

### 4.3. Text Augmentation Strategy

In order to synthesize code switched text, we utilised the prompting and ablation strategy presented in [11]. We utilise the same embedding, recurrent and batch size mentioned above. Each of the neural language models is trained for 30 epochs and used to synthesize text at the end of each epoch. The synthetic text is used to train an n-gram language model, therefore we train 30 separate n-gram language models. We utilise this approach because, in previous research, we found that our networks synthesize higher quality text when training continues after the development set loss has converged. This convergence typically occurred after 5 to 10 epochs. We then select the n-gram language model which affords the greatest reduction in development set perplexity among the 30 optimised n-grams trained on the text generated after each training epoch. This model is then utilised for lattice generation.

The augmentation strategy for one n-gram language model entailed synthesising text in batches of 100,000 sequences. Synthesis continues until the development set perplexity reductions achieved by a n-gram trained on $n$ pooled batches of synthesized text and interpolated with the n-gram trained on the training set improves by less than 0.1% relative to the interpolated n-gram trained on the previous $n-1$ batches. The language model interpolation weight is optimised on the development set perplexity after each batch.

## 5. RESULTS: ARCHITECTURAL OPTIMISATION

We optimise the model presented in Figure 1 in a greedy fashion according to the possible variations outlined in Table 2. We begin by considering the unoptimised model $\mathbb{B}$. Then, for each row in the table, an alternative configuration was evaluated and either selected or not based on whether the average development set perplexity ($PP_{Dev}$) or the associated code-switched perplexity ($CPP_{Dev}$) were improved over the four language pairs. The results of this process are presented in Table 3. The corresponding word error rates for the best performing systems, obtained by n-best rescoring are shown in Table 4. In the table, code-switched bigram error (CSBG) is calculated by measuring the average number of correctly identified words immediately following a language switch. We note that improving speech recognition has been found by other researchers to be extremely challenging for this dataset [9].

**Table 3**. Development set results for the considered configurations during the greedy optimization of the code predictive network, outlined in Section 5. PP: Overall perplexity. CPP: Perplexity only over code-switches. EZ: English-isiZulu, EX: English-isiXhosa, ES: English-Sesotho, and ET: English-Setswana.

| | EZ | | EX | | ES | | ET | |
|---|---|---|---|---|---|---|---|---|
| Alias | PP | CPP | PP | CPP | PP | CPP | PP | CPP |
| $\mathbb{A}$ | 976.6 | 5228.8 | 888.1 | 8048.9 | 350.3 | 3209.7 | 204.9 | 1476.1 |
| $\mathbb{B}$ | 1017.6 | 6490.2 | 762.1 | 15501.6 | 336.9 | 3173.1 | 206.1 | 1554.7 |
| $\mathbb{C}$ | 981.7 | 6349.6 | 744.7 | 15472.6 | 331.9 | 3538.6 | 203.7 | 1581.8 |
| $\mathbb{D}$ | 956.1 | 5498.4 | 738.6 | 12284.8 | 327.8 | 2958.4 | 198.7 | 1567.9 |
| $\mathbb{E}$ | 904.8 | 5147.1 | 744.2 | 12077.1 | 322.5 | 2449.7 | 199.3 | 1260.6 |
| $\mathbb{F}$ | 920.2 | 5225.3 | 760.0 | 13480.0 | 332.0 | 2454.5 | 198.8 | 1293.0 |
| $\mathbb{G}$ | **832.1** | 4841.4 | **725.4** | 13198.6 | **319.1** | 2728.6 | **194.2** | 1358.3 |

Our unoptimised code predictive model $\mathbb{B}$ performs worse in terms of development set perplexity for isiZulu and Setswana than the baseline LSTM ($\mathbb{A}$). However, a small improvement is seen in perplexity for Sesotho, and a relative improvement of 14.2% is achieved for isiXhosa.

Our first alternative model ($\mathbb{C}$) to the unoptimised model $\mathbb{B}$, shares the embedding layer between the two LSTM language models and the code predictor LSTM. This improves perplexity across all four language pairs compared to $\mathbb{B}$, and is therefore adopted for incorporation into our architecture. In addition, this architecture reduces the model size considerably.

The next architecture ($\mathbb{D}$) is based on model $\mathbb{C}$ but shares the hidden and cell state vectors between the two language model LSTMs and the code predictor LSTMs. This is achieved by first passing the vectors to the language predictor, whose output state vectors are fed to the language models as input. This afforded improvements in both overall perplexity and code-switched perplexity on the development set across all four language pairs, and was therefore incorporated into the model.

Architecture $\mathbb{E}$ utilises the architecture $\mathbb{D}$ and additionally applies a soft interpolation of the likelihood vectors at the output of the two monolingual language models. Table 3 shows

**Table 4**. 50-best rescoring results for best performing models on the test set. WER is word error rate %, and CSBG is the code-switched bigram error rate %. EZ: English-isiZulu, EX: English-isiXhosa, ES: English-Sesotho, and ET: English-Setswana.

| | EZ | | EX | | ES | | ET | |
|---|---|---|---|---|---|---|---|---|
| Alias | WER | CSBG | WER | CSBG | WER | CSBG | WER | CSBG |
| $\mathbb{A}_\phi$ | 41.8 | 63.7 | 42.9 | 69.5 | 50.6 | 67.2 | 41.9 | 57.4 |
| $\mathbb{A}$ | 40.5 | 61.5 | 42.5 | 68.7 | 50.0 | 66.8 | 40.6 | 55.0 |
| $\mathbb{E}$ | **40.3** | **61.1** | 42.5 | 69.3 | **49.7** | 67.5 | **39.6** | **53.2** |
| $\mathbb{G}$ | 40.7 | 61.4 | **42.4** | **68.6** | 49.8 | **66.5** | 40.7 | 55.4 |

that this further improves development set code-switched perplexity for all four language pairs. An improvement in overall perplexity is also seen for isiZulu and Sesotho. However small regressions are seen for isiXhosa and Setswana. We incorporate this operation into our language model due to the improvements in code-switched perplexity. We note that, except for the code-switched perplexity of isiXhosa, $\mathbb{E}$ outperforms the baseline LSTM ($\mathbb{A}$) in all four language pairs (by between 1.6%-23.7% relative). As shown in Table 4, this models produces the best test set recognition accuracy for isiZulu, with an absolute improvement of 1.5% compared the the baseline $\mathbb{A}_\phi$. The best test set speech recognition and code-switched recognition is also achieved for Setswana, with absolute improvements of 2.3% and 4.2% compared to baseline $\mathbb{A}_\phi$.

Incorporating a soft interpolation of the hidden and cell state vectors ($\mathbb{F}$) at the output of the two monolingual language models into architecture $\mathbb{E}$ leads to regressions in terms of overall perplexity and code-switched perplexity of 10 and 380 respectively on average over the four language pairs compared to model $\mathbb{E}$. Therefore, our model maintains the hard selection of state vectors.

In architecture $\mathbb{G}$, we incorporate a language embedding into architecture $\mathbb{E}$. The language embedding is concatenated with the word embedding and presented to the language predictor and language models as input. This variant achieves improvements in development set perplexity between 5.2%-18.3% relative to the baseline LSTM ($\mathbb{A}$). However, code-switched perplexity deteriorates for three of the language pairs (except isiZulu) compared to model $\mathbb{E}$. Therefore, we utilise both models $\mathbb{E}$ and $\mathbb{G}$ for the experiments in Section 6.

## 6. RESULTS: AUGMENTATION

Finally, we explore whether the best performing architectural variations highlighted in Table 4 are able to generate synthetic code-switched text to further improve speech recognition performance. We hypothesize that the augmented n-gram models may be able to improve the n-best hypotheses produced for rescoring. The results from the combination of both n-best rescoring and augmentation strategies are given in Table 5. Specifically, this table presents the results for the two optimised code-predictive models ($\mathbb{E}$, $\mathbb{G}$), the baseline models ($\mathbb{A}_\phi$, $\mathbb{A}$), and the intermediate code-predictive model ($\mathbb{D}$).

For isiZulu, Sesotho and Setswana, the optimised configuration $\mathbb{G}$ achieves the best word error rate, with an absolute improvement of 1.9%, 1.7%, and 2.6% respectively compared to the baseline $\mathbb{A}_\phi$, and 0.4%, 0.8%, and 0.8% respectively compared to the LSTM baseline $\mathbb{A}$. Additionally, absolute improvements in the recognition at code-switches of 3.7%, 0.9%, and 4.3% compared to baseline $\mathbb{A}_\phi$ are achieved for the same three languages respectively.

In contrast to the initial rescoring experiments in Table 4, where $\mathbb{E}$ outperformed $\mathbb{G}$ in speech recognition for isiZulu, Sesotho, and Setswana, the performance of $\mathbb{G}$ surpasses $\mathbb{E}$ when both augmentation and n-best rescoring are applied. Overall, we find that the combination of augmentation and n-best rescoring affords improvements in recognition accuracy over the baseline for all four language pairs. Furthermore, in all cases except isiXhosa, the performance was the best achieved by any architecture variation considered. On the basis of these consistent improvements across all four language pairs in both language modelling performance and speech recognition, we conclude that the inclusion of a neural component to model code-switches is a successful strategy.

**Table 5**. Test set speech recognition results for combination of augmentation and n-best rescoring. WER is word error rate %, and CSBG is the code-switched bigram error rate %. EZ: English-isiZulu, EX: English-isiXhosa, ES: English-Sesotho, and ET: English-Setswana.

| | EZ | | EX | | ES | | ET | |
|---|---|---|---|---|---|---|---|---|
| Alias | WER | CSBG | WER | CSBG | WER | CSBG | WER | CSBG |
| $\mathbb{A}_\phi$ | 41.8 | 63.7 | 42.9 | 69.5 | 50.6 | 67.2 | 41.9 | 57.4 |
| $\mathbb{A}$ | 40.3 | 60.3 | 42.6 | 69.6 | 49.7 | **66.2** | 40.1 | 53.2 |
| $\mathbb{D}$ | 40.7 | 61.7 | 42.8 | 70.9 | 50.6 | 68.7 | 39.6 | 53.7 |
| $\mathbb{E}$ | 40.3 | 61.1 | **42.0** | **68.3** | 49.0 | 66.5 | 39.8 | **52.7** |
| $\mathbb{G}$ | **39.9** | **60.0** | 42.1 | 69.3 | **48.9** | 66.3 | **39.3** | 53.1 |

## 7. CONCLUSIONS

This paper presents a new LSTM language model architecture which incorporates an additional and specific neural structure to explicitly model code-switches. We show that our best model is able to improve overall word error rates by between 0.5%-1.2% absolute across four language pairs compared to a baseline system. Additionally, we showed improvements in recognition performance specifically at code-switches of between 0.6%-2.3% absolute compared to the same baseline. Finally, we find that using these models to synthesize code-switched text that is in turn used to augment the training set of n-gram language models prior to n-best decoding and rescoring produces further improvements in recognition accuracy of 0.8%-2.6% absolute compared to the baseline. Overall, we observe that the code predictive language model affords consistent improvements across all four language pairs in both speech recognition accuracy as well as language modelling

perplexity. These consistent improvements indicate that the inclusion of a specific component to model language transitions is a successful strategy for the language modelling of code-switched speech. In future work we aim to evaluate further architectural variations of the code predictive component.

## 8. ACKNOWLEDGEMENTS

This research was supported by the Department of Sports, Arts and Culture of the Republic of South Africa. We would also like to thank the South African Centre for High Performance Computing (CHPC) for providing computational resources on their Lengau cluster for this research.

## 9. REFERENCES

[1] Jesse Emond, Bhuvana Ramabhadran, Brian Roark, Pedro Moreno, and Min Ma, "Transliteration based approaches to improve code-switched speech recognition performance," in *Proc. IEEE Spoken Language Technology Workshop (SLT)*, Athens, Greece, 2018.

[2] Min Ma, Bhuvana Ramabhadran, Jesse Emond, Andrew Rosenberg, and Fadi Biadsy, "Comparison of data augmentation and adaptation strategies for code-switched automatic speech recognition," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, United Kingdom, 2019.

[3] Samuel Thomas, Kartik Audhkhasi, and Brian Kingsbury, "Transliteration based data augmentation for training multilingual ASR acoustic models in low resource settings," in *Proc. Interspeech*, Shanghai, China, 2020.

[4] Grandee Lee, Xianghu Yue, and Haizhou Li, "Linguistically motivated parallel data augmentation for code-switch language modeling," in *Proc. Interspeech*, Graz, Austria, 2019.

[5] Karan Taneja, Satarupa Guha, Preethi Jyothi, and Basil Abraham, "Exploiting monolingual speech corpora for code-mixed speech recognition," in *Proc. Interspeech*, Graz, Austria, 2019.

[6] Heike Adel, Ngoc Thang Vu, Franziska Kraus, Tim Schlippe, Haizhou Li, and Tanja Schultz, "Recurrent neural network language modeling for code switching conversational speech," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, Canada, 2013.

[7] Saurabh Garg, Tanmay Parekh, and Preethi Jyothi, "Code-switched language models using dual RNNs and same-source pretraining," in *Proc. Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Brussels, Belgium, 2018.

[8] Emre Yilmaz, Henk van den Heuvel, and David van Leeuwen, "Acoustic and textual data augmentation for improved ASR of code-switching speech," in *Proc. Interspeech*, Hyderabad, India, 2018.

[9] Astik Biswas, Emre Yilmaz, Febe de Wet, Ewald van der Westhuizen, and Thomas Niesler, "Semi-supervised development of ASR systems for multilingual code-switched speech in under-resourced languages," in *Proc. 12th Language Resources and Evaluation Conference (LREC)*, Marseille, France, 2020.

[10] Atsunori Ogawa, Naohiro Tawara, and Marc Delcroix, "Language model data augmentation based on text domain transfer," in *Proc. Interspeech*, Shanghai, China, 2020.

[11] Joshua Jansen van Vueren and Thomas Niesler, "Optimised code-switched language model data augmentation in four under-resourced South African languages," in *Proc. SPECOM*, St. Petersburg, Russia, 2021.

[12] Ching-Ting Chang, Shun-Po Chuang, and Hung-Yi Lee, "Code-switching sentence generation by generative adversarial networks and its application to data augmentation," in *Proc. Interspeech*, Graz, Austria, 2019.

[13] Bidisha Samanta, Sharmila Reddy, Hussain Jagirdar, Niloy Ganguly, and Soumen Chakrabarti, "A deep generative model for code switched text," in *Proc. 28th International Joint Conference on Artificial Intelligence (IJCAI)*, Macao, 2019.

[14] Ishan Tarunesh, Syamantak Kumar, and Preethi Jyothi, "From machine translation to code-switching: Generating high-quality code-switched text," *arXiv preprint arXiv:2107.06483*, 2021.

[15] Yingying Gao, Junlan Feng, Ying Liu, Leijing Hou, Xin Pan, and Yong Ma, "Code-switching sentence generation by BERT and generative adversarial networks," in *Proc. Interspeech*, Graz, Austria, 2019.

[16] Carol Myers-Scotton, *Duelling languages: Grammatical structure in codeswitching*, Oxford University Press, 1997.

[17] Shana Poplack, "Sometimes I'll start a sentence in Spanish y termino en español: Toward a typology of code-switching," *The bilingualism reader*, vol. 18, no. 2, pp. 221–256, 2000.

[18] Xinhui Hu, Qi Zhang, Lei Yang, Binbin Gu, and Xinkang Xu, "Data augmentation for code-switch language modeling by fusing multiple text generation methods," in *Proc. Interspeech*, Shanghai, China, 2020.

[19] Ewald van der Westhuizen and Thomas Niesler, "Automatic speech recognition of English-isiZulu code-switched speech from South African soap operas," *Procedia Computer Science*, vol. 81, pp. 121 – 127, 2016.

[20] Ewald van der Westhuizen and Thomas Niesler, "A first South African corpus of multilingual code-switched soap opera speech," in *Proc. Eleventh International Conference on Language Resources and Evaluation (LREC)*, Miyazaki, Japan, 2018.

[21] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," in *Advances in Neural Information Processing Systems*, Virtual, 2020.

[22] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely, "The Kaldi speech recognition toolkit," in *Proc. IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, Hawaii, USA, 2011.

[23] Andreas Stolcke, "SRILM-an extensible language modeling toolkit," in *Proc. Seventh International Conference on Spoken Language Processing (ICSLP)*, Colorado, USA, 2002.