GCM: A Toolkit for Generating Synthetic Code-mixed Text

Mohd Sanad Zaki Rizvi Anirudh Srinivasan Tanuja Ganu
Monojit Choudhury Sunayana Sitaram

Microsoft Research India
{v-mori, t-ansrin, tanuja.ganu, monojitc, sunayana.sitaram}@microsoft.com

Abstract

Code-mixing is common in multilingual communities around the world, and processing it is challenging due to the lack of labeled and unlabeled data. We describe a tool that can automatically generate code-mixed data given parallel data in two languages. We implement two linguistic theories of code-mixing, the Equivalence Constraint theory and the Matrix Language theory to generate all possible code-mixed sentences in the language-pair, followed by sampling of the generated data to generate natural code-mixed sentences. The toolkit provides three modes: a batch mode, an interactive library mode and a web-interface to address the needs of researchers, linguists and language experts. The toolkit can be used to generate unlabeled text data for pre-trained models, as well as visualize linguistic theories of code-mixing. We plan to release the toolkit as open source and extend it by adding more implementations of linguistic theories, visualization techniques and better sampling techniques. We expect that the release of this toolkit will help facilitate more research in code-mixing in diverse language pairs. [1] [2]

1 Introduction

Code-mixing, which is the alternation between two or more languages in a single conversation or utterance is prevalent in multilingual communities all over the world. Processing code-mixed language is challenging due to the lack of labeled as well as unlabeled data available for training NLP models. Since code-mixing is a spoken language phenomenon, it is more likely to occur in informal written text, such as social media and chat data. Such data may not as easily available as monolingual data for building models, and may also exhibit other issues such as cross-transcription and non-standard spellings.

To alleviate this problem and train language models that can use unlabeled data for pre-training, we see the generation of synthetic code-mixed data as a promising direction. Various linguistic theories have been proposed that can determine how languages are mixed together, and in prior work we presented the first computational implementation (Bhat et al., 2016) of the Matrix-language (Myers-Scotton, 1993) and Equivalence Constraint theories (Poplack, 1980). We also showed that generating synthetic data using our computational implementations improved word embeddings leading to better downstream performance on sentiment analysis and POS tagging (Pratapa et al., 2018b), as well as RNN language models (Pratapa et al., 2018a). The multilingual BERT (Devlin et al., 2019) model fine-tuned with synthetic code-mixed data outperformed all prior techniques on the GLUECoS benchmark (Khanuja et al., 2020) for code-switching, which spans 11 NLP tasks in two language pairs. The approach of generating synthetic code-mixed data has gained traction following our work, with other approaches including using Generative Adversarial Networks (Chang et al., 2019), an encoder-decoder framework with transfer learning (Gupta et al., 2020), using parallel data with a small amount of real code-mixed data to learn code-mixing patterns (Winata et al., 2019) and a novel two-level variational autoencoder approach (Samanta et al., 2019).

In this work, we present a tool GCM that can automatically generate synthetic code-mixed data given parallel data or a Machine Translation system between the languages that are being mixed. Our tool is intended for use by NLP practitioners who would like to generate training data to train models that can handle code-mixing, as well as linguists and language experts who would like to visualize how code-mixing occurs between languages given

[1] Screencast: https://aka.ms/eacl21gcmdemo
[2] Code: https://aka.ms/eacl21gcmcode

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations, pages 205–211
April 19 - 23, 2021.

different linguistic theories. The toolkit provides three modes - a batch mode, that can run the data generation pipeline on servers, an interactive mode, that can be used for quick prototyping as well as a web interface that can be used to visualize code-mixed sentence generation. The GCM tool will be released as open source and we plan to improve it by adding more implementations of linguistic theories, visualization techniques and better algorithms for sampling. We expect that the release of this toolkit will spur research in code-mixing in diverse language pairs and enable many NLP applications that would not be possible to build due to the lack of code-mixed data.

2 Method

In this section we discuss the linguistic theories that we implement in the tool and the pipeline we use for generating code-mixed (hereafter referred to as CM) sentences.

2.1 Linguistic theories

Our tool currently contains implementations of two two linguistic theories for generating valid CM text: Equivalence Constraint Theory (Poplack, 1980) and Matrix Language Theory (Myers-Scotton, 1993).

The **Equivalence Constraint Theory** states that intra-sentential code-mixing can only occur at places where the surface structures of two languages map onto each other, thereby, implicitly following the grammatical rules of both the languages. The **Matrix Language Theory** deals with code-mixing by introducing the concept of "Matrix Language" or the base language into which pockets of the "Embedded Language" or second language are introduced in such a way that the former sets the grammatical structure of the sentence while the later "switches-in" at grammatically correct points of the sentence.

1E. With this the prescribed fee will be sent.
1H. इसके साथ निर्धारित शुल्क भेजा जाएगा।
Transliterated. iske saath nirdhaarith shulkh bheja jayega.

1CM. इसके साथ prescribed fee भेजा जाएगा।
Transliterated. iske saath prescribed fee bheja jayega.

2E. My husband is working on his master's degree.
1S. Mi marido está trabajando en su maestría.
2CM. Mi marido está working on his master's degree.

Figure 1: (a) Input sentences for Hindi-English (1E, 1H) and Spanish-English (2E, 1S) and (b) GCM output Code-Mixed sentences for Hindi-English (1CM) and Spanish-English (2CM).

Figure 1 shows example source sentences in Hindi, English and Spanish and their CM counterparts generated by the EC theory. Figures 2 and 3 show the parse trees of all the sentences above, illustrating how the sentences are generated by the theory.

2.2 Code-mixed (CM) Text Generation Process

The generation process is a sequential process (Figure 4), which requires parallel sentences in the two languages being mixed as input data. Three ma-

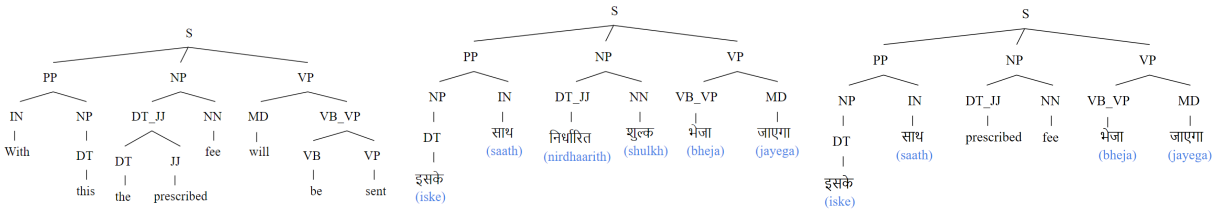

Figure 2: Parse-trees of (a) sentences [1E] and (b) [1H], and (c) of [1CM] according to the EC Theory

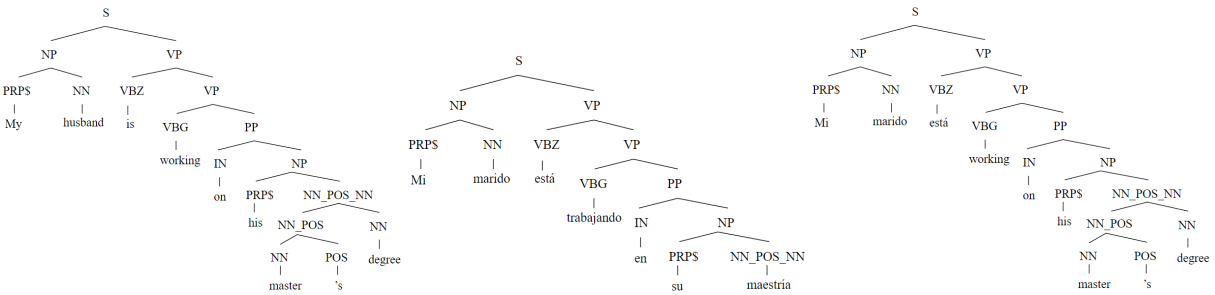

Figure 3: Parse-trees of (a) sentences [2E] and (b) [1S], and (c) of [2CM] according to the EC Theory

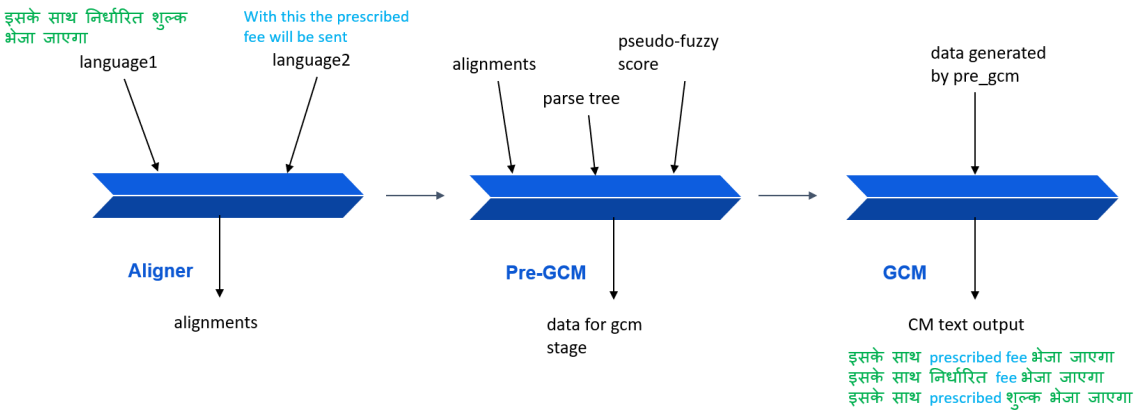

Figure 4: The CM Generation Process

jor components play a part in the process and the stages occur in the following order:

The first stage is the "Alignment stage". In this stage, the **Aligner** is used to generate word level alignments for input pair of sentences. We currently use "fast_align" (Dyer et al., 2013) which performs well compared to other aligners in terms of both speed and accuracy.

The second stage is the **Pre-GCM** stage which is responsible for pre-processing the input. This stage combines the aligner outputs along with constituent parse trees generated by the parser and "Pseudo Fuzzy-match Score" (Pratapa et al., 2018a) for each sentence pair to make one row of input data for the GCM stage. The **Parser** is used to generate a sentence level constituent parse tree for one of the source languages. Previously in (Pratapa et al., 2018a) we used the Stanford Parser (Klein and Manning, 2003) but we now also provide the option to use the Berkeley Neural Parser (Kitaev and Klein, 2018). This stage is also responsible for creating appropriate batches of data to be consumed by the next stage.

The final **GCM** stage, processes each batch of data, applying linguistic theories in order to generate CM sentences as output.

2.3 Sampling

Figure 5 shows some sentences generated by the EC theory for a pair of Hindi-English source sentences. Through manual observation and user studies, we find that the EC theory generates sentences that may be grammatically correct, but may not feel natural to bilingual speakers. In prior work we showed that sampling appropriately from the generated data is crucial. We experimented with various sampling techniques and showed that training an RNN Language Model with sampled synthetic data reduces the perplexity of the model by an amount which is equivalent to doubling the amount of real CM data available (Pratapa et al., 2018a). So, we add a sampling stage after the generation stage, for which we propose the following techniques.

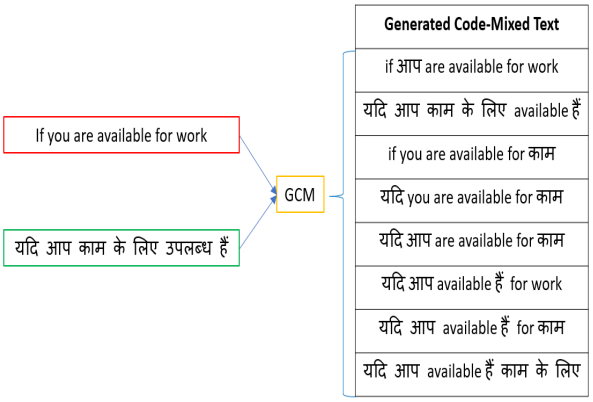

Figure 5: Need for Sampling: Not all generated CM sentences feel natural

- **Random**: For each parallel pair of input sentences, we arbitrarily pick a fixed number k of CM sentences from the generated corpus. The advantage of this method is that we are not dependent on having real CM data.
- **SPF-based**: The **Switch Point Fraction or SPF** is the number of switch points in a sentence divided by the total number of words in the sentence (Pratapa et al., 2018a). For each parallel pair of input sentences, we randomly pick k CM sentences such that the SPF distri-

bution of these is as close as possible to that of the real CM data. The benefit of this method is that we can generate a synthetic CM corpus that close to the real data distribution in terms of amount of switching, but this method imposes a requirement of having real CM data for the given language pair.

- **Linguistic Features-based**: Words do not get switched at random, and it would be useful to be able to learn patterns of switching from real CM data. For example, learning how nouns and verbs tend to get switched can create more realistic data. However, this method imposes additional requirements - in addition to real CM data, we also need POS taggers for CM data, which are not readily available.

Out of the above techniques, Random and SPF-based sampling are currently implemented in the system. In the future, we would like to add improved sampling techniques to the tool, since it is an important step to achieve high quality synthetic data.

3 System Overview

We provide three modes in the GCM tool: a batch mode, an interactive library mode and a web-interface to address the needs of NLP practitioners, researchers, linguists and language experts:

3.1 Batch Mode

This mode is primarily intended for those who want to generate CM data on servers given large parallel corpora of monolingual data. It operates via a configuration file that contains multiple options to customize CM text generation. We describe some of the options available in batch mode (Listing 1). The entire list of options can be found in the code documentation.

```
[GENERAL]
.
.
# choose which stages of the pipeline
    are going to be run; default: pregcm
    , gcm
stages_to_run =
# whether to run the pregcm and gcm
    stages parallely; default: 0 ; set
    to 1 to run parallely
parallel_run =

[ALIGNER]
.
.

[PREGCM]
.
# cut-off value for PFMS score
max_pfms =
# select the parser to be used from
    available parsers - stanford and
    benepar; default: benepar
parser =

[GCM]
.
# max number of sentences to generate
    per sentence; default: 5
k =

[OUTPUT]
# language tag at word level in each
    output code-mixed sentence
lid_ouput =
# visualize DFAs that were used to make
    generations
dfa_output =
# sampling technique to use - random or
    spf
sampling =
```

Listing 1: Options available in the configuration file in batch mode

In the `[GENERAL]` section, the option `stages_to_run` lets the user choose specific stages to be run on the data. When a large scale CM corpus is to be generated, it is useful to run the CM generator pipeline in parallel mode to speed up the process. The `parallel_run` options lets the user run the Pre-GCM and GCM stages asynchronously so that instead of waiting for all the data to be pre-processed, the GCM stage can start working on batch of data as and when ready.

The `max_pfms` option in `[PREGCM]` lets user select the "Pseudo Fuzzy-match Score" threshold for the input sentences. In order to prepare consistent input data, we perform back-translation as one of the steps. The Pseudo Fuzzy-match Score quantifies the quality of back-translation that directly impacts the quality of CM data generated, hence this feature is particularly important.

`parser` lets you choose between the Stanford Parser and Berkeley Natural Parser. The Stanford Parser contains support for parsing Arabic, Chinese, English, French, German and Spanish, while the Berkeley Natural Parser can parse English, Chinese, Arabic, German, Basque, French, Hebrew, Hungarian, Korean, Polish, Swedish. While we rely on one of these supported languages to be one of the two languages in the parallel corpus from which the CM text is generated, we generate the second parse tree using the alignments from the previous step. So, we can generate CM sentences

in language pairs where one of the languages is supported by either of the two parsers.

The `k` option in `[GCM]` controls the maximum number of CM sentences to be generated per input sentence. Similarly, the `lid_output` and `dfa_output` options in the `[OUTPUT]` lets the user extract additional information in the form of word-level language tags and DFAs for each generated CM sentence. This can be used for debugging the CM generation process, since the user can see the language tags assigned to the generated CM sentence in case both languages are in the same script. The sampling option lets the user choose the kind of sampling technique they want for generating CM text: currently, the options available are Random or SPF based, as described earlier.

3.2 Library Mode

The library mode is a light weight interactive interface for a programmer to go back and forth with the output of various stages to adjust parameters. This mode was designed to be able to accommodate modules that the user may want to add to the pipeline to increase speed, accuracy and language coverage. The library is designed to be continuously extensible, for example, to add a new preprocessing sub-module or a parser in a language that the available parsers do not support. Below is an example of using the library mode to experiment with CM generation by utilizing the outputs of both the Stanford Parser and the Berkeley Neural Parser (Listing 2):

```
from gcm.aligners import fast_align
from gcm.parsers import benepar,
    stparser
from gcm.stages import pregcm, gcm


# code to generate alignments using
    fast_align
# assuming corpus is the variable
    storing data

aligns = fast_align.gen_aligns(corpus)

# code to use benepar to generate parse
    trees from the corpus
pt_benepar = benepar.parse(corpus)

# code to use stanford parser to
    generate parse trees from the corpus
pt_stanford = stparser.gen_parse(corpus)

# generating two set of CMs one based on
     the stanford parser and the other
    on benepar
# assuming pfms_scores to have PFS of
    the input sentences

pgcm_benepar = pregcm.process(corpus,
    aligns, pt_benepar)
pgcm_stanford = pregcm.process(corpus,
    aligns, pt_stanford)

gcm_stanford = gcm.gen(pgcm_stanford)
gcm_benepar = gcm.gen(pgcm_benepar)

# now both the generated CMs can be
    trained on downstream language-
    modeling tasks to compare their
    performance
```

Listing 2: Using library mode to generate CM text based on Benepar and Stanford Parser parse trees.

3.3 Web UI

In addition to the batch mode and library modes, which are targeted at users who want to either create large amounts of CM data or are proficient programmers, we also wanted to create a way for linguists and language experts to be able to visualize linguistic theories of code-mixing in an intuitive and easy to use interface. For this, we created a Web UI mode, which we describe next. The Web UI mode is meant to generate CM sentences for one pair of input sentences at a time.

The user can provide either a pair of parallel sentences, or can use the Translate option to translate a source sentence into another language using translation APIs. The user can choose the linguistic theory that they want to use to generate the CM text as can be seen in Figure 6.

Once the user has selected the options and clicks on the generate button, we generate the output of GCM which consists of all the parse trees and the generated CM sentences. As shown in Figure 7, we show all possible sentences generated by the linguistic theory and do not restrict the number of sentences or sample them. This is to enable users to see all the sentences that are generated by the linguistic theory, which can then be restricted or sampled by using the code in batch or library mode. We expect that the Web UI will be very useful as the support for more implementations of CM theories increases, as well as to visualize CM between different language pairs.

4 Conclusion and Future Work

Generating synthetic CM data has become a promising direction in research on code-mixing, due to the lack of available data and has proved to be successful in improving various CM NLP tasks.

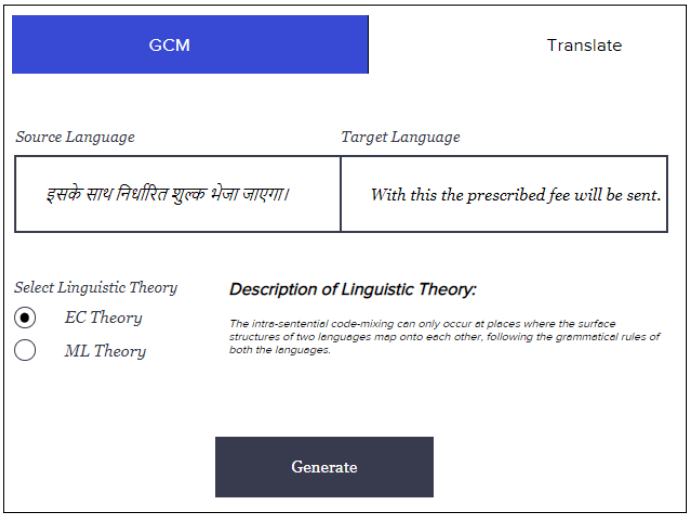

Figure 6: Selecting the linguistic theory and giving input sentences to GCM Web UI.

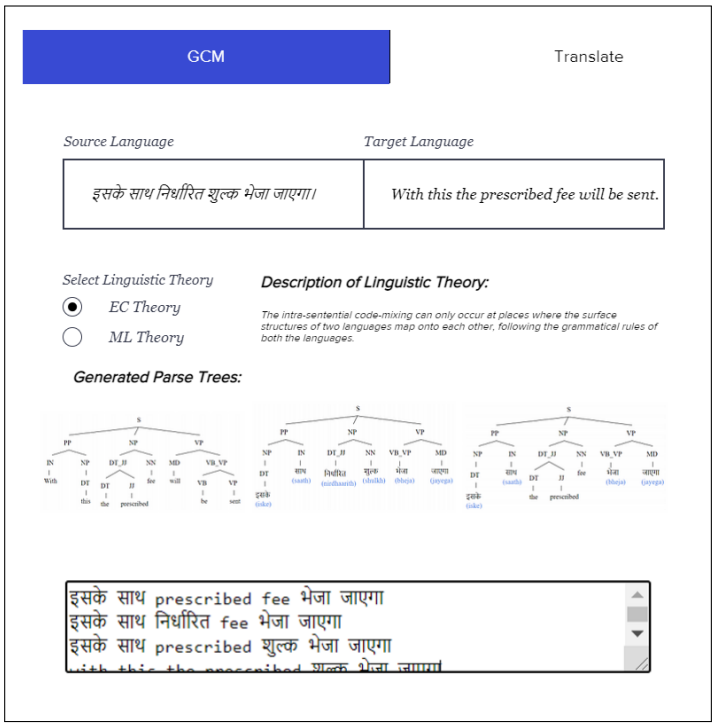

Figure 7: Code-Mixed sentences and the associated parse trees as output.

In this paper, we describe a tool for generating synthetic CM data given parallel data in two languages, or a translator between two languages. We implement two theories of code-mixing, the Equivalence Constraint (EC) theory and the Matrix-Language (ML) theory to generate CM data, followed by a sampling stage to sample sentences that are close to real code-mixing in naturalness. The GCM tool operates in three modes - a batch mode, which is meant for large scale generation of data, a library mode, which is meant to be customizable and extensible and a Web UI, which is meant as a visualization tool for linguists and language experts.

We plan to release the GCM tool as open source code and add more implementations of linguistic theories, generation techniques and sampling techniques. We believe that this tool will help address some of the problems of data scarcity in CM languages, as well as help evaluate linguistic theories for different language pairs and we expect that the release of this toolkit will spur research in diverse code-mixed language pairs.

References

Gayatri Bhat, Monojit Choudhury, and Kalika Bali. 2016. Grammatical constraints on intra-sentential code-switching: From theories to working models. *arXiv preprint arXiv:1612.04538*.

Ching-Ting Chang, Shun-Po Chuang, and Hung-Yi Lee. 2019. Code-switching sentence generation by generative adversarial networks and its application to data augmentation. *Proc. Interspeech 2019*, pages 554–558.

J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A simple, fast, and effective reparameterization of IBM model 2. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–648, Atlanta, Georgia. Association for Computational Linguistics.

Deepak Gupta, Asif Ekbal, and Pushpak Bhattacharyya. 2020. A semi-supervised approach to generate the code-mixed text using pre-trained encoder and transfer learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 2267–2280.

Simran Khanuja, Sandipan Dandapat, Anirudh Srinivasan, Sunayana Sitaram, and Monojit Choudhury. 2020. Gluecos: An evaluation benchmark for code-switched nlp.

Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Melbourne, Australia. Association for Computational Linguistics.

Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Carol Myers-Scotton. 1993. Duelling languages: Grammatical structure in code-switching. *Clarendon Press, Oxford.*

Shana Poplack. 1980. Sometimes i'll start a sentence in spanish y termino en espaÑol: toward a typology of code-switching 1. *Linguistics*, 18:581–618.

Adithya Pratapa, Gayatri Bhat, Monojit Choudhury, Sunayana Sitaram, Sandipan Dandapat, and Kalika Bali. 2018a. Language modeling for code-mixing: The role of linguistic theory based synthetic data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1543–1553.

Adithya Pratapa, Monojit Choudhury, and Sunayana Sitaram. 2018b. Word embeddings for code-mixed language processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3067–3072.

Bidisha Samanta, Sharmila Reddy, Hussain Jagirdar, Niloy Ganguly, and Soumen Chakrabarti. 2019. A deep generative model for code-switched text. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5175–5181. AAAI Press.

Genta Indra Winata, Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. 2019. Code-switched language models using neural based synthetic data from parallel sentences. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 271–280.