grade: 93

WebNLG Challenge 2019

Alexander Malis
acm503@psu.edu
Department of Computer Science and Engineering,
Pennsylvania State University
State College, PA

Bodhisatwa Chatterjee
bxc583@psu.edu
Department of Computer Science and Engineering,
Pennsylvania State University
State College, PA

Alexandar Devic
amd6185@psu.edu
Department of Computer Science and Engineering,
Pennsylvania State University
State College, PA

Sree Sai Teja Lanka
szl577@psu.edu
Department of Computer Science and Engineering,
Pennsylvania State University
State College, PA

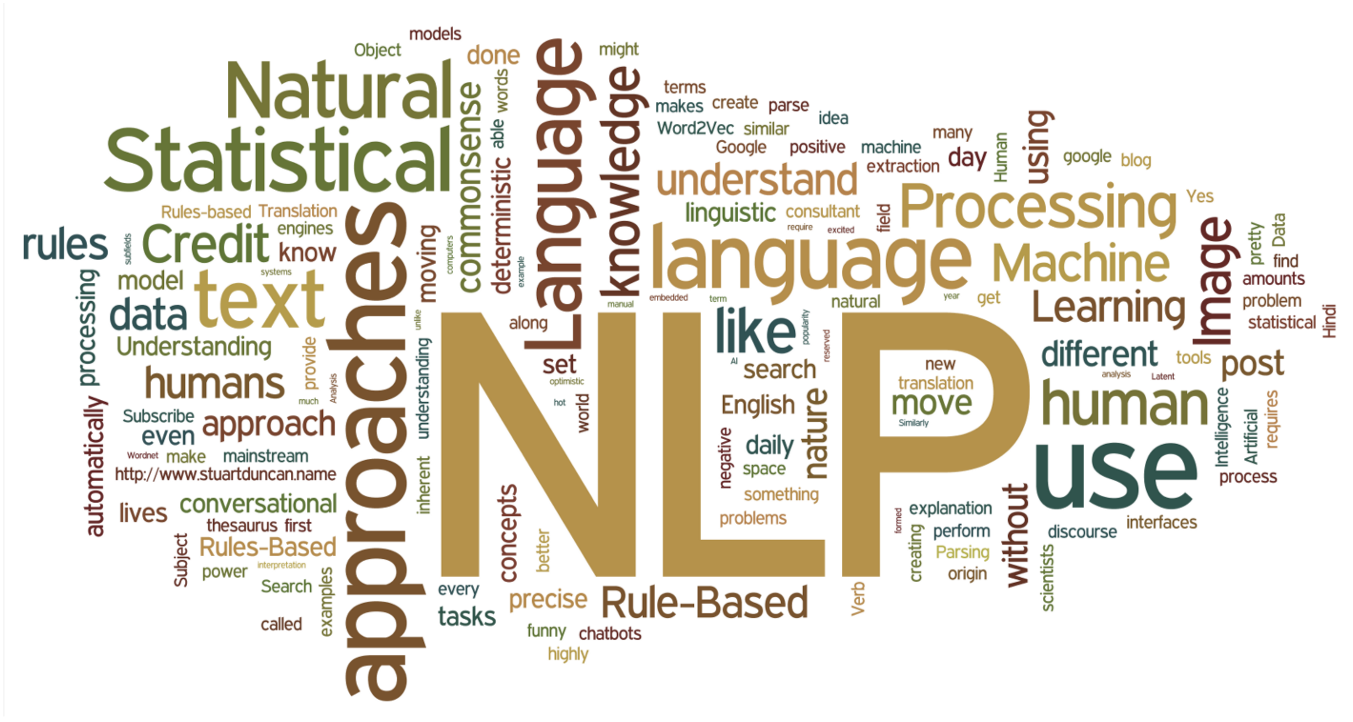


Figure 1: Natural Language Processing Graphic

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

ABSTRACT

This report is a continuation of Phase 1 and is designed to provide competitive BLEU scores to a limited Web NLG challenge. This challenge was applied with the TensorFlow NMT model and run both on a local system as well as on the ICS-ACI network. The role of this project is to act as a foundation for future learning and research into NMT. Learning objectives were as follows: *ICS-ACI usage, NMT optimization and usage, and general research practices.* This phase was further broken down into three parts, A, B and C. In A, a wide range of preprocessing, vocab and data splits

were tested. Phase B uses the search space reduction from phase A to test different variations of a nmt model. Phase C is when the pieces from A and B are put together to achieve the best BLEU score possible. C

CCS CONCEPTS

• **Artificial Intelligence** → **Natural Language Processing**; • **Natural Language Processing** → *Natural Language Generation*; • **Natural Language Generation** → Neural Machine Translation.

KEYWORDS

Neural Machine Translation, Recurrent Neural Networks, Attention-Based Models, Delexicalization

ACM Reference Format:
Alexander Malis, Bodhisatwa Chatterjee, Alexandar Devic, and Sree Sai Teja Lanka. 2019. WebNLG Challenge 2019. In *CSE 597 Spring'19: Natural Language Processing, April 21, 2019, State College, PA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION Intro: 5/5

The goal of machine translation is to produce translation outputs by taking input sentences from a human language. However, natural language generation takes in a input of semantic representation of a structured semantic language and gives the output of human language sentences. One example of such a structured semantic language is the RDF triple. The motive of WebNLG challenge is to take in RDF triples as its input and generate English texts as its output. It can be considered as a mapping or translation from the RDF triple language to the textual English language [4]. The RDF triples are taken from *Dbpedia*[2]. Neural Machine Translation(NMT) is the model which we will use for this task. The way NMT works is that it takes in the entire sentence, focuses on encoding/decoding using one or more hidden layers derived from co-occurrence aspects of meaning, based on sequences of tokens and then produces a translation. NMT model follows an encoder-decoder architecture for achieve this task. The *encoder* takes in the English sentence and transforms it into a vector[sequence of numbers]. Then it uses the *decoder* to generate the translation. [3]

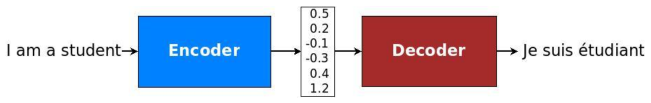


Figure 2: Encoder-decoder architecture: Example of a general approach for NMT. An encoder converts a source sentence into a "meaning" vector which is passed through a decoder to produce a translation.[5]

This challenge was applied with the TensorFlow NMT model and run both on a local system as well as on the ICS-ACI network. The role of this project is to act as a foundation for future learning and research into NMT. Learning objectives were as follows: *ICS-ACI usage, NMT optimization and usage, and general research practices.* This phase was further broken down into three parts, A, B and C. In A, a wide range of preprocessing, vocab and data splits were tested. Phase B uses the search space reduction from phase A to test different variations of a NMT model. Phase C is when the pieces from A and B are put together to achieve the best BLEU scored model possible. Ultimately, a model with an unseen test data BLEU score of 46.2 was achieved which places first on the submissions from the original WebNLG challenge in the unseen test data category. Data prep: 18/20
Some of the differences between methods seem pretty minor

2 DATA PREPARATION

Data preparation refers to both vocab and preprocessing. For this section of the project, there are three primary phases: **A, B** and **C**. In A, a wide range of data preparation was tested and compared to find a clear path forward. Data from Phase A lead to the training data and models selected, as a conglomeration of ideas tried by the four individual members.

Vocab Preparation

It can be argued that the initial conditions and data quality fed to our model impact its performance a great deal. However, it was unclear at first what method of vocabulary generation would both yield the best results and be easily applicable to further test data. Because of this, a vocab generator with five different parse methods were created for filtering of English sentences.

- **SPACE SPLIT (Method 0)** - The simplest of English word extraction, each sentence was split on each space and the words gathered would be added to the vocab set. For example, the English sentence: *Alan Bean, a crew member of NASA's Apollo 12, was.born March 15, 1932.* would contain vocabulary words *{"Alan", "Bean,", "a", "crew", "member", "of", "NASA's", "Apollo", "12,", "was.born", "March", "15,", "1932."}* Notice however, punctuation, possessives, and errors within the sentence carry through with the vocabulary generation.
- **NO PUNCT (Method 1)** - Stepping up from **Method 0**, this method does the same as before however removes all punctuation from the vocabulary words. With the running example, the English sentence: *Alan Bean, a crew member of NASA's Apollo 12, was.born March 15, 1932.* would now contain vocabulary words *{"Alan", "Bean", "a", "crew", "member", "of", "NASA's", "Apollo", "12", "was.born", "March", "15", "1932"}*

Fixes to the data would be a great thing to submit to the WebNLG site, and to me for future use. Would you be able to send me corrected data?

We did not cover the topic of word tokenization in class, but FYI a word tokenizer would change {"NASA's"} to {"NASA", "'s"}

- **NO PUNCT POSSESSIVES (Method 2)** - This method builds on top from **Method 1** while removing all possessives and some types of contractions. With this, the English sentence: *Alan Bean, a crew member of NASA's Apollo 12, was.born March 15, 1932.* would now contain vocabulary words *{"Alan", "Bean", "a", "crew", "member", "of", "NASA", "Apollo", "12", "was.born", "March", "15", "1932"}*
- **NO PUNCT FIX (Method 3)** - A recurring theme in the WebNLG corpus of training data was malformed English sentences. We feared that these errors were numerous and could negatively impact our model in its inference testing, having nothing to do with the model but just by having bad data. To combat these concerns, a recursive algorithm was created to attempt to fix and extract words in malformed sentences. This method was then added on top of **Method 1**, dubbed with a "fix" tag in its method name. Using the same example: *Alan Bean, a crew member of NASA's Apollo 12, was.born March 15, 1932.* now contain vocabulary words *{"Alan", "Bean", "a", "crew", "member", "of", "NASA's", "Apollo", "12", "was", "born", "March", "15", "1932"}* Notice that this parse and extraction method yields the most accurate English vocab extraction solely based on the input sentence.
- **NO PUNCT POSSESSIVES FIX (Method 4)** - The last method builds on **Method 3** in a similar manner to how **Method 2** built on top of **Method 1**. With this: *Alan Bean, a crew member of NASA's Apollo 12, was.born March 15, 1932.* now contain vocabulary words *{"Alan", "Bean", "a", "crew", "member", "of", "NASA", "Apollo", "12", "was", "born", "March", "15", "1932"}*

For RDF sentences, a simple "space-pipe-space" split was used to extract RDF words. Since there was no room for error in this extraction method, no other method was designed for RDF vocab extraction.

With the vocabulary files built off of feeding through all the lex sentences and RDF sentences, multiple pairs of vocab files were saved for later use by the model. The generator also includes an extractor method which, given an unsanitary sentence, produces a sanitary sentence using one of the methods above. This sanitizer was used across all sentences before they were run through the model in training, testing, and inferencing.

what do you mean by generator? the decoder?

Non-delexicalized Preparation

For Phase A, 4 types of non-delexicalized parsing were chosen to test. First is a direct mapping of the processed data as a control with no modifications. Next two methods are ordering and compression. With one method featuring each as a stand alone model, and one with them joined to create a multi-step parse strategy. To cover these methods, consider the standard three triple input and two lex entries.

This is confusing. What is the principle behind the method?

[VI_1] $A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I$	[EN_1] Lex_1
[VI_2] $A \mid B \mid C \mid D \mid E \mid F \mid G \mid H \mid I$	[EN_2] Lex_2

Ordering based processing attempts to order related triples to produce a streamlined output. This works on a simple concept that if you have two triples $A|B|C$ and $D|E|F$ and F = A then the order should be changed to $D|E|F$ and $A|B|C$. This places the input closer to a usable form. For the standard triple with $A = I$, and $D = I$, the ordering based parsing is as follows:

[VI_1] $G \mid H \mid I \mid D \mid E \mid F \mid A \mid B \mid C$	[EN_1] Lex_1
[VI_2] $G \mid H \mid I \mid D \mid E \mid F \mid A \mid B \mid C$	[EN_2] Lex_2

Because of implementation, $A \mid B \mid C$ will be placed behind $G \mid H \mid I$, and then $D \mid E \mid F$ will also be placed behind $G \mid H \mid I$. This method results in the same number of triples and Lex entries. No space is saved with this method and no input size reductions are achieved.

Compression based preprocessing is designed to compact multiple triples ~~into each other~~ when possible. This works by taking triples where 2 of the 3 elements are shared. Conceptually, we can think of this is something like a transformation from two triples '*Alexander | likes | cats*' and '*Alexander | likes | dogs*' to a single triple '*Alexander | likes | cats and dogs*'. This folds *cats* and *dogs* together and splits them with *"and"*. This results in a reduced number of triples which reduces the complexity of the problem. No space is saved in this form of pre-processing. When $A = D = G$ and $B = E = H$ in standard entry:

[VI-1] $A \mid B \mid C$ and F and I	[EN-1] Lex-1
[VI-2] $A \mid B \mid C$ and F and I	[EN-2] Lex-2

This approach **does not** do anything to lower the number of line entries but it does compress the problem for processing in. When both compression and ordering were performed on input, this resulted in a *levenshtein distance*[8] of 56 per 1000 characters on sample input. This shows the small scope of the changes done to the data.

After testing on a standard NMT model, both ordering and compression perform drastically inferior to the standard parse. As such, for all models after the standard model they were not included as a time saving measure. For comparison, unprocessed data had a minimum BLEU score on unseen data of **18.4**. The max score from the other 3 parsing methods was **2.1**. The difference is immense, resulting in the models being dropped from our further evaluation.

In Phase A, testing with these models produced results with high dev scores; however, this was a result of over fitting. Dev scores quickly diverge from test scores as the model learns to fit the data, resulting in the discrepancy.

Do you mean the levenshtein distance of the original to the processed version? is 56 the average?

Moving forward to Phase B, no preprocessing aside from the basic case was tested and used.

Did you think about using any of the methods used in the challenge?

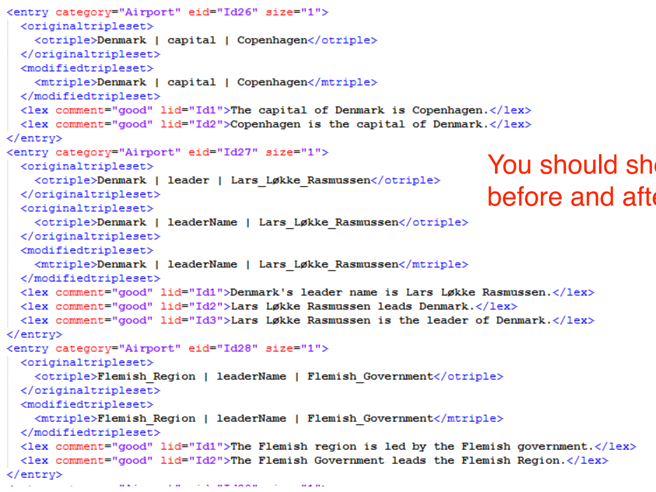
Delexicalized Preparation

To make sure that our model is able to generalize the 'context' of the word in a better sense, we have used delexicalization for Phase C. We have defined the 'context' of a word as the specific category to which it belongs. For example, in a triple like *'Bob | loves | America'*, the word 'America' falls in the category of Countries, while the word 'Bob' comes in the category of 'Name'. By delexicalization, we mean that we have replaced a specific word, present in both the triple and the lex comment, and replaced it with its corresponding category. The reason for incorporating such a modification is to achieve generalization while training the model.

Although the idea of delexicalization seems simple enough, the task is non-trivial. To achieve complete delexicalization, we need to associate words present in both lex comments and RDF triples with their respective categories. The task of defining categories for words is subjective. A word like 'Bob' can be categorized as 'Name' or as 'Proper Noun'. We came up with an approach which we call an '*incremental approach*' for applying delexicalization in our data. We have first identified the 'popular' categories present in our data (raw xml files) by carefully studying the data. By popular categories, we mean the categories which would cover the maximum number of words in itself. In this project, we handled the following categories of words for the delexicalization:

- **Countries** and **Languages** : These were one of the most common occurring words. Separate lists of countries and languages spoken in them were built with the help of the python module *pycountry*. Any specific country occurring in either the lex comments or the RDF triple was replaced by the keyword *'COUNTRY'*. Fig 3 shows an example of delexicalizing countries.
- **Airports**: Since this was a specific category in our data, we delexicalized all names of airports occurring in our data by the keyword *'AIRPORT'*. A list of airport names were prepared from Dbpedia.
- **Astronauts**: A list of all the names of astronauts was made from Wikipedia and all the occurrences of names of astronauts were replaced by the keyword '*ASTRONAUT*'.
- **Animals**: A comprehensive list of all animals was made from *Colorado State University Libraries*[1] and all occurrences were replaced by keyword *'ANIMAL'*.

The *incremental approach* of delexicaization means that we have incorporated these categories one by one and trained our model at each step. The initial test BLEU score increased to a maximum of **45.6**, but then **started decreasing as more and more categories got added**. This was counter-intuitive and we would be discussing this in the discussion section.



```
<entry category="Airport" eid="Id26" size="1">
  <originaltripleset>
    <otriple>Denmark | capital | Copenhagen</otriple>
  </originaltripleset>
  <modifiedtripleset>
    <mtriple>Denmark | capital | Copenhagen</mtriple>
  </modifiedtripleset>
  <lex comment="good" lid="Id1">The capital of Denmark is Copenhagen.</lex>
  <lex comment="good" lid="Id2">Copenhagen is the capital of Denmark.</lex>
</entry>
<entry category="Airport" eid="Id27" size="1">
  <originaltripleset>
    <otriple>Denmark | leader | Lars_Løkke_Rasmussen</otriple>
  </originaltripleset>
  <originaltripleset>
    <otriple>Denmark | leaderName | Lars_Løkke_Rasmussen</otriple>
  </originaltripleset>
  <modifiedtripleset>
    <mtriple>Denmark | leaderName | Lars_Løkke_Rasmussen</mtriple>
  </modifiedtripleset>
  <lex comment="good" lid="Id1">Denmark's leader name is Lars Løkke Rasmussen.</lex>
  <lex comment="good" lid="Id2">Lars Løkke Rasmussen leads Denmark.</lex>
  <lex comment="good" lid="Id3">Lars Løkke Rasmussen is the leader of Denmark.</lex>
</entry>
<entry category="Airport" eid="Id28" size="1">
  <originaltripleset>
    <otriple>Flemish_Region | leaderName | Flemish_Government</otriple>
  </originaltripleset>
  <modifiedtripleset>
    <mtriple>Flemish_Region | leaderName | Flemish_Government</mtriple>
  </modifiedtripleset>
  <lex comment="good" lid="Id1">The Flemish region is led by the Flemish government.</lex>
  <lex comment="good" lid="Id2">The Flemish Government leads the Flemish Region.</lex>
</entry>
```

You should show before and after

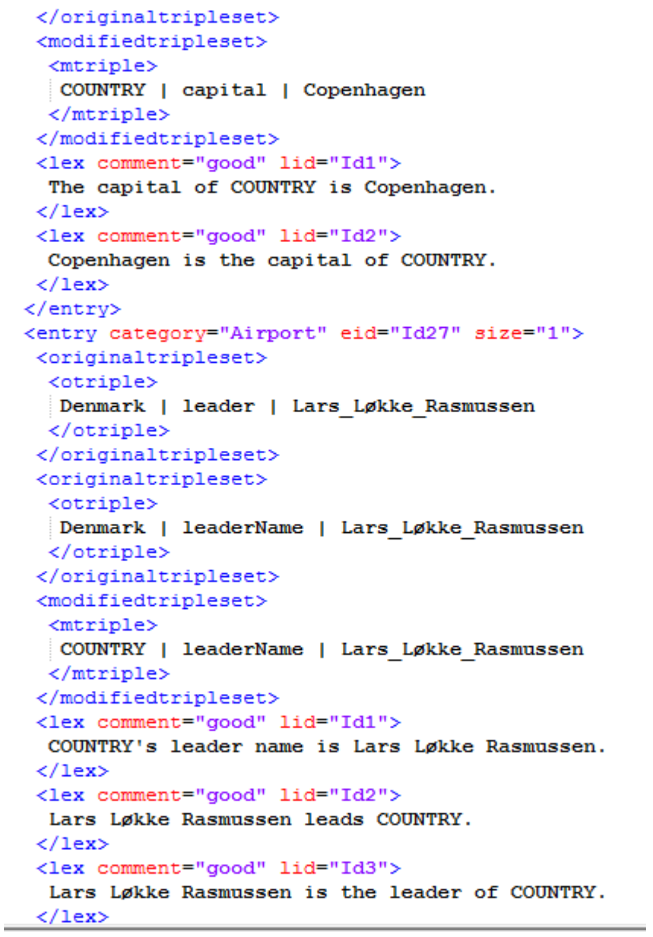
Figure 3: Sample XML input file before delexicalization of Category Country

Model & Training: 20/20

3 MODEL AND TRAINING PARAMETERS

Models were split into three different phases: A, B and C. For Phase A, a standard model was used and the data input was varied. In Phase B, numerous models were tested to refine and illuminate a path forward to success. In Phase C, a key model was selected and optimized for success. All results will be shown in results section aside from test data for the competing model components.

- **Standard Model** - 12000 steps, 100, steps per stats, 2 layers of 256 units, dropout of 0.2 training on BLEU metrics with a greedy inference mode. - A simple model listed in the tutorial for TensorFlow/NMT was used as the control model. This is a known good example and was selected to show whether future models tested were performing well.
- **Beam Model** - 12000 steps, 100, steps per stats, 2 layers of 256 units, dropout of 0.2 training on BLEU metrics with a beam search inference mode and a beam width of 12. - A standard model, modified to include a beam search. Beam search replaces the traditionally greedy approach and gives the model a limited ability to look at future states before deciding on what course of action to take. If this problem was to be related to chess, and one move had a high probability of success but the next move had drastically reduced chances it would give the model the ability to select a different path before the bad decision. This is intended to sacrifice performance in exchange for a more efficient



```
  </originaltripleset>
  <modifiedtripleset>
   <mtriple>
    COUNTRY | capital | Copenhagen
   </mtriple>
  </modifiedtripleset>
  <lex comment="good" lid="Id1">
   The capital of COUNTRY is Copenhagen.
  </lex>
  <lex comment="good" lid="Id2">
   Copenhagen is the capital of COUNTRY.
  </lex>
 </entry>
 <entry category="Airport" eid="Id27" size="1">
  <originaltripleset>
   <otriple>
    Denmark | leader | Lars_Løkke_Rasmussen
   </otriple>
  </originaltripleset>
  <originaltripleset>
   <otriple>
    Denmark | leaderName | Lars_Løkke_Rasmussen
   </otriple>
  </originaltripleset>
  <modifiedtripleset>
   <mtriple>
    COUNTRY | leaderName | Lars_Løkke_Rasmussen
   </mtriple>
  </modifiedtripleset>
  <lex comment="good" lid="Id1">
   COUNTRY's leader name is Lars Løkke Rasmussen.
  </lex>
  <lex comment="good" lid="Id2">
   Lars Løkke Rasmussen leads COUNTRY.
  </lex>
  <lex comment="good" lid="Id3">
   Lars Løkke Rasmussen is the leader of COUNTRY.
  </lex>
```

Figure 4: Sample XML input file after delexicalization of Category Country(we are only replacing the modified triples)

model. Run speed was not measured for this report but the performance cost was not noticeable to testing staff as compared to the standard model.

- **Attention Model** - 12000 steps, 100, steps per stats, 2 layers of 256 units, dropout of 0.2 training on BLEU metrics with an attention based model. Pilot experiments showed scaled_luong to be the most effective and it was what was used for this model. Attention based models have extra weights added to pair input and output data that makes sure all aspects of the input are accounted for in the output and that it is not counted multiple times. This provides a reduction to duplications and a solution to missed inclusions from the input data. As was mentioned, run speed was not measured for this report however this time the performance cost was noticeable to testing staff as compared to the standard model.[6]

good

you should say why

- **Variable parameters** - Three parameters of the original model were varied, each one as a power of two from the previous. These are: *Steps* from 12,000 to 24,000, *layers* from 2 to 4 and *units* from 256 to 512. These were varied at exactly double to gauge the effectiveness for comparable run time costs. Despite this, adding layers had the lowest cost, runs was second and units were the most costly. These numbers were only measured roughly while running models so the scale of the difference is unclear.

This resulted in a list of tiers of models to test. Tier 0 was the standard model, and the number of parameters changed on a subsequent model from the standard, results in its tier. This gives tiers 0 to 5. Tier 0 and 1 were tested with 5 vocab and 2 splits, resulting in 10 instances per model. Subsequent models were tested with 2 splits and 1 vocab. This results in a 10x6 data set and a 2x32 data set.

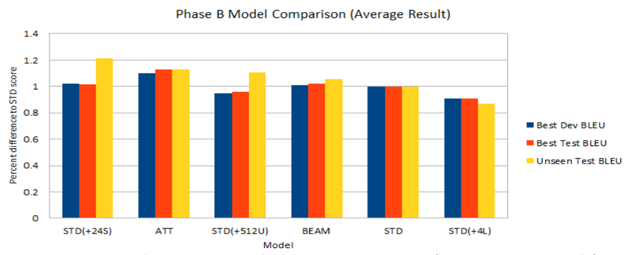


Figure 5: Phase B Model Comparison (Average result)

Key	%	Model (Avg)	Dev Perplexity	Best Dev BLEU	Test Perplexity	Best Test BLEU	Unseen Test
1st		STD(+24S)	109.16%	102.36%	110.34%	101.52%	121.09%
2nd		ATT	91.41%	109.77%	91.80%	112.92%	112.75%
3rd		STD(+512U)	123.37%	94.77%	123.23%	95.79%	110.90%
4th		BEAM	99.98%	100.79%	99.48%	102.25%	105.78%
5th		STD	100.00%	100.00%	100.00%	100.00%	100.00%
6th		STD(+4L)	109.53%	90.83%	113.35%	90.80%	87.02%
	Raw	Model (Avg)	Dev Perplexity	Best Dev BLEU	Test Perplexity	Best Test BLEU	Unseen Test
		STD(+24S)	4.994	22.1	5.528	20.74	30.78
		ATT	4.182	23.7	4.599	23.07	28.66
		STD(+512U)	5.644	20.46	6.174	19.57	28.19
		BEAM	4.574	21.76	4.984	20.89	26.89
		STD	4.575	21.59	5.01	20.43	25.42
		STD(+4L)	5.011	19.61	5.679	18.55	22.12

Table 1: Phase B Model Comparison (Colored table)

In Phase C, a final architecture was selected based on data from Phases A and B. This resulting model was a standard model with beam search, attention, 24,000 steps and 512 units. Four layers were not used due to their negative influence on BLEU scores. This is the model used and combined with lexicalized parsing to give the best result. When only the model is considered, this gives a BLEU score of 46.2 using the simple parse. This is compared to the simple model used in Phase A which resulted in a max score of 22.8. The final

result is over double the starting point.
When comparing the results of each style of model in Tier 1 to the control model in Tier 0:

- **STD with 24,000 steps** - Increasing the number steps by a factor of two increases the run time by a factor of two. It also increases a risk of over fitting. Despite this, if over fitting is managed, this produces the best results from the Tier 1 models tested. There is a dependency between the dev and testing scores and the ultimate unseen test score. When doing analysis of the translations, the model appears likely to answer related topics with a stored answer. For example, *"Alexander | likes | cats"* and *"Alexander | likes | dogs"* can sometimes be confused in the model to over fit and answer what it knows, not what it is working with. Additional steps need to be taken in order to prevent over fitting. For future testing, a higher dropout is recommended.
- **ATT** - This model did comparable to all testing data with a flat increase to scores. This is indicative of attention based models, when comparing to the problem of STD with 24,000 steps, this model limits the chance to be mislead by previous test data. By forcing attention, *"Alexander | likes | cats"* and *"Alexander | likes | dogs"* will almost always produce different results because the attention model is pulling the result to include each part of the input.
- **STD with 512 units** - With more units, the problems from STD with 24,000 steps are increased. The data is more likely to fit and predict itself on the training data. When these hints are removed with unseen data, the model proves to have increased results. Additional steps need to be taken in order to prevent over fitting. For future testing, a higher dropout is recommended.
- **BEAM** - This model offer slight improvements over the greedy model. This was below expectations but in line with the training data. As the input data is more condensed, the model is making less choices with higher weight then if it was for example translating Vietnamese to English. This means that there are less opportunities for beam search to optimize, as the input step generates multiple English words at a time. Beam search will do best when a later word changes the meaning of a previous word. This however, is not as common in triple format.
- **STD with 4 layers** - This is the only model with consistently negative results. Using 4 layers is not recommended unless a model can be produced that has a need for it. Even with this, it would likely be best to have layers with different functions, rather then multiple layers of the same function. This could be viable with longer training but this was not explored in this paper. This is a focus for future research as in theory, layers should be able to produce positive results in the correct settings.

4 TEST/TRAIN SPLIT Test/Train 5/5

The train/test split was evaluated over the sub-phases of the project, A, B and C. In Phase A, two splits were considered; one of a dev/train/test weighted at 10/80/10 and a second at 10/85/5. These splits were compared over the course of the Phase A testing, compromising different vocab, parsing, and models to find the optimum balance. These values were selected because of prior results in the four phase 1 papers of the group members. This data split was handled by iterating over training examples, and maintaining a count of modulo 20, with specific counts going to different sources. This allowed for easy shifting of data in 5% increments. This consisted of 3 and 7 being assigned to dev, 11 and 13 being assigned to test and the remaining assigned to train. B consisted of 2 and 17 being assigned to dev, and 5 being assigned to test with the remaining numbers assigned to train. This ensures that A.Dev, A.Test, B.Dev and B.Test remain independent.

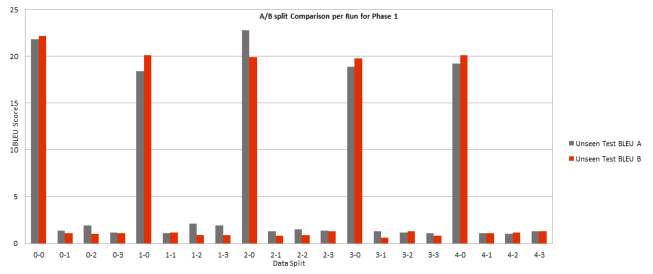


Figure 6: Phase A - A/B split comparison per run

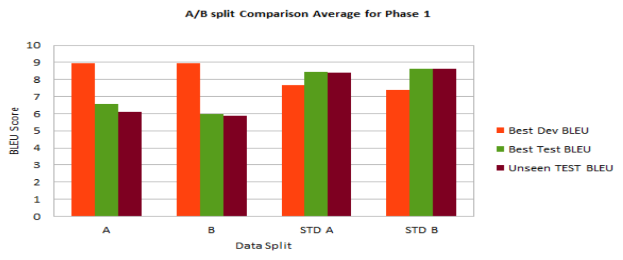


Figure 7: Phase A - A/B split comparison average

From Phase A, the results were too close to limit to a distinct winner. This pushed the phase evaluation back to Phase B of the project.

In Phase B, split B had a slight advantage over split A. Because the best two models were progressed forward, A

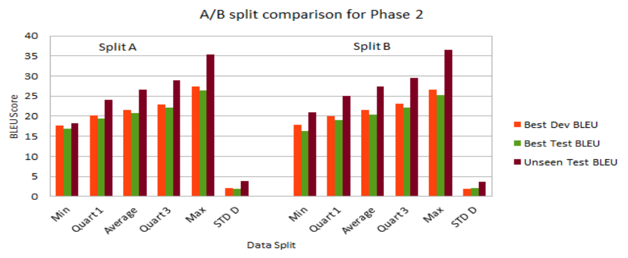


Figure 8: Phase B - A/B split comparison average

and B both continued to Phase C as A vocab 0 and B vocab 0 were the top scoring models.

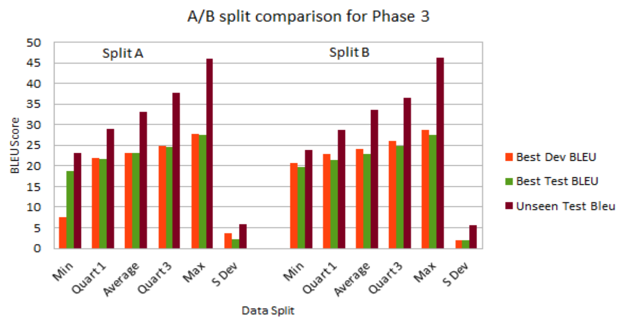


Figure 9: Phase C - A/B split comparison average

With the final comparison between split A and B, we can draw some conclusions. First, split B produces better results on average. This is believed to be from the extra test data making a more robust model when it came to unseen data. This is a small improvement and is likely the result of having slightly more training data. For future testing, it is recommended to have a fixed dev and training section, and modify only the testing data. This would remove the penalty that split A had less training data then split B and give clearer results on the effect of the split on the model.

5 RESULTS

Results: 20/20 very thorough

Results and training grades are calculated using BLEU scores. BLEU stands for **Bilingual Evaluation Understudy**[7]. BLEU scores are calculated by taking the nmt output and comparing it to a target sentence. A higher score shows a result that is closer to the desired output. This is done by checking the output words and checking if they exist in the desired output with a modified precision. Words are counted up the maximum occurrence in the target sentence. This prevents duplication from scoring highly in the model. In standard precision, a single word generated multiple times as output would score highly, if that word was in the target sentence, and that is not the desired output.

you should know what perplexity is; it's a standard language model score; it was presented in the 20190117-jurafskyLM slides

In addition to BLEU scores, perplexity is a score given by tensorflow/NMT that represents a difficulty of a problem. Lower scores represent less difficult problems. While training, this number generally decreases which reflects the model growing in capability.

The following data is the results from Phases A, B and C showing the progression of results:

Name	Dev Perplexity	Best Dev BLEU	Test Perplexity	Best Test BLEU	Unseen Test BLEU
STD-0-0-A	5.58	25	6.05	23.4	21.8
STD-0-1-A	7.36	5.1	9.55	2	1.4
STD-0-2-A	7.03	4.7	12.05	1.9	1.9
STD-0-3-A	7.35	5	10.53	1.8	1.2
STD-1-0-A	4.58	21.1	4.97	20.3	18.4
STD-1-1-A	6.06	4.5	8.15	1.8	1.1
STD-1-2-A	5.6	4.4	8.18	1.8	2.1
STD-1-3-A	5.71	5	7.24	1.6	1.9
STD-2-0-A	4.28	21.2	4.64	20.4	22.8
STD-2-1-A	6.4	4.5	12.38	1.9	1.3
STD-2-2-A	5.59	4.7	8.43	1.8	1.5
STD-2-3-A	6.02	4.5	8.14	1.7	1.4
STD-3-0-A	4.3	20.9	4.6	19.9	18.9
STD-3-1-A	5.84	4.9	8.03	2	1.3
STD-3-2-A	5.53	4.7	7.73	1.7	1.2
STD-3-3-A	5.29	4.4	8.47	1.9	1.1
STD-4-0-A	4.41	20.7	4.81	19.9	19.2
STD-4-1-A	5.46	4.3	8.3	1.9	1.1
STD-4-2-A	5.44	4.7	7.57	1.9	1
STD-4-3-A	5.36	4.5	8	1.9	1.3
STD-0-0-B	5.52	23.5	6	22.2	22.2
STD-0-1-B	6.77	4.7	11.32	1	1.1
STD-0-2-B	6.74	5	11.54	1.4	1
STD-0-3-B	6.81	4.9	12.92	1	1.1
STD-1-0-B	4.28	21	4.71	19.8	20.1
STD-1-1-B	5.48	4.5	9.84	1.2	1.2
STD-1-2-B	5.96	5.5	10.45	1.2	0.9
STD-1-3-B	5.37	5.1	10.08	1.1	0.9
STD-2-0-B	4.4	21	4.96	20.2	19.9
STD-2-1-B	5.95	4.8	8.77	0.9	0.8
STD-2-2-B	5.77	4.4	11.03	0.9	0.9
STD-2-3-B	5.43	4.7	10.92	1.4	1.3
STD-3-0-B	4.31	20.6	4.79	20.3	19.8
STD-3-1-B	5.94	4.9	8.65	0.9	0.6
STD-3-2-B	5.33	5	9.64	1.1	1.3
STD-3-3-B	4.69	4.7	8.03	1.3	0.8
STD-4-0-B	4.29	21	4.69	20.2	20.1
STD-4-1-B	5.58	4.4	10.78	1.2	1.1
STD-4-2-B	5.76	4.9	8.91	1.1	1.2
STD-4-3-B	5.46	4.4	11.58	1.2	1.3

Table 2: Phase A Data

Does the corresponding perplexity tell you anything? Why report it?

Table 2 shows our results on the Phase A data. The ranges of Bleu scores achieved varied drastically from a low of **0.6** to a high of **22.2** on the unseen test data.

Table 3 shows our results on the Phase B data. The ranges of Bleu scores achieved increased significantly with a low of **18.2** to a high of **33.7**.

Table 4 shows our results on the Phase C non-delexicalized data. This is where our Bleu scores on the unseen test data performs the best. As we can see, we have a high of **46.2** with a low of **22.3**.

Table 5 shows our result on the Phase C delexicalized data on the best performing model from the last phase. As we could observe, the best performing model in Table 4 was the one with Beam Search, Attention-Based Mechanism, 24k runs and 512 Unit using Split B and Vocab 0. This was the model on which the delexicalized data was trained on. The best score attained on the unseen data was **45.6** with only one category of delexicalization (*Language and Countries*). The lowest score on unseen data is **31** where all 4 categories were included in delexicalization.

Index (Model)	Best Dev BLEU	Best Test BLEU	Unseen Test BLEU	Model	Steps
ATT-12000-256-2	27.4	26.5	33.7	ATT	12000
ATT-12000-256-2	26.6	25.2	33.6	ATT	12000
ATT-12000-256-2	23.4	22.5	29.5	ATT	12000
ATT-12000-256-2	22.3	21.9	28.4	ATT	12000
ATT-12000-256-2	22.3	21.9	25.9	ATT	12000
ATT-12000-256-2	23	22.2	27.6	ATT	12000
ATT-12000-256-2	24.1	23.1	26	ATT	12000
ATT-12000-256-2	23.6	22.4	28.7	ATT	12000
ATT-12000-256-2	21.1	21.7	23.9	ATT	12000
ATT-12000-256-2	23.2	23.3	29.3	ATT	12000
BEAM-12000-256-2	23.8	23.1	29.1	BEAM	12000
BEAM-12000-256-2	23.7	22.1	29.7	BEAM	12000
BEAM-12000-256-2	21.5	21.1	27.1	BEAM	12000
BEAM-12000-256-2	20.6	19.7	26.3	BEAM	12000
BEAM-12000-256-2	21.2	20.9	25.8	BEAM	12000
BEAM-12000-256-2	21.7	20.8	26.5	BEAM	12000
BEAM-12000-256-2	21.2	20.9	26.2	BEAM	12000
BEAM-12000-256-2	21.7	19.4	25.8	BEAM	12000
BEAM-12000-256-2	20.8	20.4	25.9	BEAM	12000
BEAM-12000-256-2	21.4	20.5	26.5	BEAM	12000
STD-12000-256-2	23.9	23	31.8	STD	12000
STD-12000-256-2	24.2	22.7	30.5	STD	12000
STD-12000-256-2	21.3	20.4	24	STD	12000
STD-12000-256-2	21	19.8	24.8	STD	12000
STD-12000-256-2	21.1	20.4	24	STD	12000
STD-12000-256-2	21.4	20	24.9	STD	12000
STD-12000-256-2	20.6	19.7	24.5	STD	12000
STD-12000-256-2	21	19.4	24	STD	12000
STD-12000-256-2	20.6	19.4	22.6	STD	12000
STD-12000-256-2	20.8	19.5	23.1	STD	12000
STD-12000-256-4	23	22.3	26.5	STD	12000
STD-12000-256-4	23.1	21.4	26.5	STD	12000
STD-12000-256-4	19.3	18.7	20.7	STD	12000
STD-12000-256-4	18.9	17.4	22.1	STD	12000
STD-12000-256-4	19.2	18.1	21.5	STD	12000
STD-12000-256-4	19.2	18.1	21.6	STD	12000
STD-12000-256-4	18.5	18.1	21.2	STD	12000
STD-12000-256-4	17.9	16.3	20.9	STD	12000
STD-12000-256-4	17.6	16.9	18.2	STD	12000
STD-12000-256-4	19.4	18.2	22	STD	12000
STD-12000-512-2	23.1	22.6	32.6	STD	12000
STD-12000-512-2	23.5	22.1	33.7	STD	12000
STD-12000-512-2	19.7	19.3	27.3	STD	12000
STD-12000-512-2	19.6	18.8	27.3	STD	12000
STD-12000-512-2	19.6	19.1	26.8	STD	12000
STD-12000-512-2	19.8	18.6	27.6	STD	12000
STD-12000-512-2	19.9	19.5	26.1	STD	12000
STD-12000-512-2	19.8	18.4	28.1	STD	12000
STD-12000-512-2	20	18.9	26.3	STD	12000
STD-12000-512-2	19.6	18.4	26.1	STD	12000
STD-24000-256-2	24.7	23.3	35.4	STD	24000
STD-24000-256-2	24.9	23.3	36.5	STD	24000
STD-24000-256-2	21.5	20.6	29.8	STD	24000
STD-24000-256-2	21.6	20.4	30.2	STD	24000
STD-24000-256-2	21.3	20.7	28.3	STD	24000
STD-24000-256-2	21.2	19.7	30.5	STD	24000
-24000-256-2	21.7	20.3	28.9	STD	24000
STD-24000-256-2	21.7	19.5	30.1	STD	24000
STD-24000-256-2	21.1	19.9	28.7	STD	24000
STD-24000-256-2	21.3	19.7	29.4	STD	24000

Table 3: Phase B Data

Index	Dev Perplexity	Best Dev BLEU	Test Perplexity	Best Test BLEU	Unseen Test BLEU
BEAT-24000-512-2-B	6.32	26.5	7.5	25.9	46.2
BEAT-24000-512-2-A	8.11	27.3	9.19	26.8	46
ATT-24000-512-2-B	6.94	26.2	8.42	24.7	45.4
ATT-24000-512-2-A	4.82	7.54	8.74	26.4	45.2
BEAT-24000-256-2-B	5.72	28.7	6.41	27.6	41.4
ATT-24000-256-2-B	6.1	26.9	6.94	26.5	40.4
BEAT-24000-256-2-A	5.28	27.8	5.79	27.5	40.3
BEAT-24000-512-4-A	11.89	24.1	13.73	23.7	40.1
ATT-24000-256-2-A	4.47	27.5	7.27	26.8	39.6
ATT-12000-512-2-B	5.69	26.9	6.56	25.8	38.7
BEAM-24000-512-2	8.21	23	9.08	21.7	38.7
BEAM-24000-512-2-A	8.17	22.9	9.03	22.2	38.4
STD-24000-512-2-A	7.89	23.5	8.65	23.3	38.3
STD-24000-512-2-B	8.18	23.3	9.17	22.7	38.3
ATT-24000-256-4-A	7.39	25.8	8.26	24.6	37.6
ATT-24000-512-4-B	9.1	23.6	11.38	22.8	36.5
BEAM-24000-256-2-B	5.21	25	6.2	23.1	36.5
STD-24000-256-2-B	6.04	24.9	6.81	23.3	36.5
ATT-12000-512-2-A	6.05	26	6.88	25.3	36.1
ATT-24000-256-4-B	6.18	26	7.01	25.2	36
BEAM-24000-256-2-A	5.76	24.9	6.3	23.5	35.7
BEAT-24000-256-4-B	6.31	25.7	7.42	24.5	35.7
STD-24000-256–A	6.02	24.7	6.57	23.3	35.4
BEAM-24000-512-4-B	9.29	22.2	11.66	20.4	35.2
BEAT-12000-512-2-B	5.61	26	6.64	25.1	35.2
BEAT-24000-512-4-B	9.34	23.4	10.98	22.5	34.8
BEAM-12000-512-2-B	6.97	22.4	7.68	21.1	34.2
ATT-12000-256-2-A	4.68	27.4	5.1	26.5	33.7
STD-12000-512-2-B	7.05	23.5	7.85	22.1	33.7
BEAT-12000-256-2-B	5.32	26.1	5.96	25.9	33.6
ATT-12000-256-2-A	5.06	26.6	5.82	25.2	33.6
BEAT-12000-256-2-B	5.45	26	6.33	24.8	33.5
BEAM-24000-256-4-B	6.77	24.8	8.99	22.8	33.1
BEAT-12000-512-2-A	5.46	24.7	6.09	24.5	32.8
STD-12000-512-2-A	6.68	23.1	7.27	22.6	32.6
BEAM-12000-512-2-A	6.85	21.8	7.45	21.4	31.8
STD-12000-256-2-A	5.25	23.9	6.13	23	31.8
BEAT-24000-256-4-A	6.44	25.4	7.14	24.6	31.5
BEAM-24000-256-4-A	6.6	24.5	7.58	24.1	31
STD-24000-512-4-A	9.38	21.8	10.71	21.2	30.8
BEAM-24000-512-4-A	7.4	21.2	8.39	21.2	30.6
ATT-24000-512-4-A	10.14	22.6	11.34	21.7	30.5
STD-12000-256-2-B	5.59	24.2	6.2	22.7	30.5
STD-24000-512-4-B	7.4	21.5	9.15	20	30.5
BEAM-12000-256-2-B	5.31	23.7	5.79	22.1	29.7
BEAM-12000-256-2-A	5.32	23.8	5.72	23.1	29.1
ATT-12000-256-4-B	5.5	24.3	6.41	23.4	29
STD-24000-256-4-A	6.74	23	7.72	21.9	28.8
-12000-256-4-B	6.02	23.5	7.48	22.6	28.3
STD-12000-512-4-A	7.62	21.5	8.67	20.9	28.1
STD-24000-256-4-B	6.66	23.1	7.97	22.2	28
STD-12000-512-4-B	7.22	22.2	8.65	20.4	27.5
BEAM-12000-512-4-B	7.18	21.2	8.63	19.8	27.4
BEAM-12000-256-4-A	6.35	23.3	7.15	23	26.5
STD-12000-256-4-A	5.98	23	6.84	22.3	26.5
STD-12000-256-4-B	5.95	23.1	7.32	21.4	26.5
BEAT-12000-256-4-A	6.23	21.6	6.68	21.1	26.4
ATT-12000-512-4-A	6.73	22.2	7.65	21.2	26.1
ATT-12000-512-4-B	6.42	21.3	7.59	21.3	25.4
BEAT-12000-256-4-B	5.68	22.2	6.31	20.8	25.3
BEAM-12000-512-4-A	8.31	19.7	9.3	18.7	24.9
BEAT-12000-512-4-B	6.18	20.8	7.23	19.8	23.8
BEAT-12000-512-4-A	6.33	20.2	7.01	20.5	23.2
ATT-12000-256-4-A	5.72	21.7	6.34	21.1	22.3

Table 4: Phase C Data, Non-delexicalized

Index	Dev Perplexity	Best Dev BLEU	Test Perplexity	Best Test BLEU	Unseen Test BLEU
DELEX-1-BEAT-24000-512-2-B	5.65	26.1	6.78	25.1	45.6
DELEX-2-BEAT-24000-512-2-B	6.58	28.6	8.79	27.5	44.9
DELEX-3-BEAT-24000-512-2-B	5.19	25.8	8.85	23.3	33.5
DELEX-4-BEAT-24000-512-2-B	4.9	25.1	7.82	22.4	31

Table 5: Phase C Data, Delexicalized

When compared to the WebNLG challenge results, there are two things to consider. First, our dev and test BLEU results are lower on the seen data section at position 8/9. Second, our unseen test scores are higher at position 1/9 (Table 6). This mixed result is not fully justified. There are two pending schools of thought on the difference. First, our training data was more difficult, or our unseen data was easier in comparison to the evaluation data used in the original

Team	Bleu Score
Our Model	**46.2**
MELBOURNE	45.13
TILB-SMT	44.28
PKUWRITER	39.88
UPF-FORGE	38.65
TILB-PIPELINE	35.29
TILB-NMT	34.60
BASELINE	33.24
ADAPT	31.06
UIT-VNU	7.07

Table 6: Comparison of Overall Test Bleu Scores with 9 Participating teams with our best model

The language in this paragraph is very unclear. I don't understand it.

WebNLG challenge. Second, there is an issue with the data split or BLEU score generation system. The second option is unlikely, because the unseen data was known as a good data set and the BLEU score metrics in tensorflow/NMT are not shown to be erroneous. Because the updated set of data was used, it is presumed to be harder and thus depressed the BLEU scores for dev and test data. This "Forged in fire" approach is likely one reason the easier unseen data performed above expectations.

Going forward, a standardized data set is needed with an unseen set that matches the difficulty of the original data set. This will produce more uniform results across testing.

6 DISCUSSION

For vocab processing, there are several takeaways from our approach. The BLEU score metric is incomplete in that it rewards a sentence that has a high translation using the same words. When the vocab file changes it complicates the problem for the NMT. Now not only does it have to translate, it has to translate in a more difficult way because its results might be correct by a human grader but get a low BLEU score. obvious Future testing will need to modify both the input and output files. For example, if vocab splits *do* and *n't* to two words from *don't*, then the output file should also have the words in the form *don't* to show the model the desired output.

In non-delexicalized prepossessing, the results were underwhelming. That this did show was the effects of over fitting. Once a dev and test score start to diverge the data is fitting to its input. In a naive sense, with early testing showing high dev scores and low test scores, this was not apparent. Instead, more runs were added, more units were added and troubleshooting was focused on trying to fix the model while overlooking the early steps of the data processing.

In delexicalized prepossessing, the initial BLEU score increased and then suddenly started declining. The possible reason for this is that whenever we are adding more categories for delexicalization, the number of new words in our target sentence is getting reduced. As a result, the BLEU score metric ratio is also decreasing. This problem is generally solved by applying **Re-Lexicalization**, where we replace the category names by the original word after the generation. yes This has to be done by keeping track of each replaced word during delexicalization. The BLEU score should improve after applying relexicalization.

From the perspective of model development, finding quantitative scores to compare the performance of different models is a strong result. NMT models are cumbersome to train and test, this leads to intuition reducing the search space in hope of saving time. This however, means that any model that does not fit into the filter imposed by intuition is untested. By testing these features, and finding the strengths and weaknesses of each, both time and effort was saved for future work. Additionally, a path forward to bring models such as using 4 layers to a helpful state exists and was made clearer based on this paper. This gives open ended research projects to study these parameters in finer detail in the future. With more research, and finer control over the granularity, there can be more work done to compare all combinations of basic parameters and find the best results for different NLP tasks.

For train/test data splitting, this paper acts as an in depth case study on two related data splits, 10/80/10 and 10/85/5 and the results are close. For future work, many smaller combinations need to be tried and more variables need to be controlled for. This can give a more in depth look at the problem and ultimately some conclusions need to be drawn about the threshold of "too little data" and "enough data to work well". At the end of the day, more data will likely always be better. What is key is finding when more data starts to give diminishing returns and when that return falls under a threshold to where it is not cost effective. This threshold needs to be able to be computed in relation to the cost of data, and the loss expected for not having enough data.

7 CONCLUSION Conclusion: 10/10

In conclusion, Phase 2 filled a number of roles. The easy first choice is a solid understanding in NLP as it relates to the WebNLG challenge. Additionally, a solid foundation in group work, research, group research and coordination was established for all members. The act of merging four development paths to produce an improved version was a unique and helpful experience to the continuation of learning in NLP for all team members involved. This paper will serve as a basis for understanding in future NMT with a well documented list of methods that work, methods that do not work and ideas of how to change things what do not work to something that can work in the future. One of the important takeaway message from this project is that importance of

Discussion: 15/20 mostly straightforward, but confusing in places, and insubstantial points about train/test split

If the model generalizes well, and you have enough data, it should not be that sensitive to small changes in the split any particular phase in natural language generation cannot be over-emphasised. Each phase from data preprocessing to selecting model parameters is as important as the other. Another interesting observation was that we had seen from the results of original WebNLG challenge[4] that it is more likely that the NMT is good at learning a representation that generalizes within the data it has seen, but it was not good at unseen test data. However, we have seen in our investigation that NMT can also be adapted to capture long-range dependencies.

REFERENCES

[1] 2019. *Animal List A to Z.* https://lib2.colostate.edu/wildlife/atoz.php?sortby=genus_species&letter=all

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference (ISWC'07/ASWC'07).* Springer-Verlag, Berlin, Heidelberg, 722–735. http://dl.acm.org/citation.cfm?id=1785162.1785216

[3] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *CoRR* abs/1409.1259 (2014). arXiv:1409.1259 http://arxiv.org/abs/1409.1259

[4] Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG Challenge: Generating Text from RDF Data. 124–133. https://doi.org/10.18653/v1/W17-3518

[5] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. 2017. Neural Machine Translation (seq2seq) Tutorial. *https://github.com/tensorflow/nmt* (2017).

[6] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *CoRR* abs/1508.04025 (2015). arXiv:1508.04025 http://arxiv.org/abs/1508.04025

[7] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL '02).* Association for Computational Linguistics, Stroudsburg, PA, USA, 311–318. https://doi.org/10.3115/1073083.1073135

[8] L. Yujian and L. Bo. 2007. A Normalized Levenshtein Distance Metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 6 (June 2007), 1091–1095. https://doi.org/10.1109/TPAMI.2007.1078