Learning Neural Templates for Text Generation

Final Project in Database Systems

Dimitrios Roussis **Nikos Episkopos**
MSc in Data Science & Information Technologies
National Kapodistrian University of Athens
Athens, Greece
{dimrous,episko}@di.uoa.gr

ABSTRACT

In recent years, there has been a considerable advancement in neural text generation models which are able to form natural, human-readable sentences from structured data. In this project, we implemented the model described in Wiseman et al. (2018) [1], where a hidden semi-Markov model (HSMM) decoder is proposed and which is able to learn latent templates so as to provide additional interpretability and controllability in text generation. Furthermore, we implemented two additional models, one for each dataset that the authors experimented upon, and evaluated them so as to reproduce the reported results. In particular, we used the model described in Dušek and Jurčíček (2016) [2] for the E2E challenge dataset [3] and the model described in Liu et al. (2016) [4] for the Wikipedia biographies (WikiBio) dataset [5]. In order to do all these, we created a GitHub repository, which contains a Docker container that can be used to reproduce the experiments we conducted and display the results that we report here through a simple user interface (UI).

KEYWORDS

Natural Language Generation, Structured Data, Sequence to Sequence, Deep Learning, Neural Networks

1. Introduction

Natural Language Generation (NLG) is a large and quickly expanding field with a large number of applications. In this project, we focus mainly on the task of converting structured data, i.e. a table containing information about a specific entity, to a human-readable natural language utterance. More specifically, the data used in such a task usually consist of a collection of records from a knowledge base which forms a **meaning representation** (MR) and its **corresponding human reference** (ref), i.e. a "friendlier" textual description.

Recent generator models used in NLG are usually recurrent neural networks (RNNs) and follow an **encoder-decoder framework**, in which the model first encodes a table into a sequence and the decoder outputs a textual description in a natural language [6]. These systems, however, tend to work like "black boxes", lacking interpretability and controllability once trained; an enduring and recurring problem in the wider field of deep learning and neural networks. The model that we mainly focused on in this project consists of a neural hidden semi-Markov model (HSMM) decoder which aims to extract discrete, template-like structures within the data which can be used for generating text while alleviating the aforementioned problems to a certain degree [1]. This model is described in section *2.1.* and will hereafter be referred to simply as **"NTG"**.

In the original paper, the authors experimented on two datasets, the E2E dataset [3] and the WikiBio dataset [5] (which are described in more detail in sections *3.1.* and *3.2.* respectively), and compared the performance of their model with other published encoder-decoder models, as well as with direct template-style baselines [1]. In this project, due to time constraints and ease of use (GitHub repositories with sufficient documentation), we implemented and evaluated two additional models, one for each dataset. For the E2E dataset, we trained the neural language model described in Dušek and Jurčíček (2016) [2] (the baseline of the E2E challenge) and evaluated it along with the model in Wiseman et al. (2018) [1] using the following metrics: BLEU [7], NIST [8], ROUGE-L [9], CIDEr [10] and METEOR [11]. The aforementioned model is described in section *2.2.* and will hereafter be referred to as **"TGen"**. As regards the WikiBio dataset, we trained the model described in Liu et al.

(2016) [4] and evaluated it (unfortunately, we were not able to get valid results from the NTG on the WikiBio dataset) using BLEU [7] and ROUGE-4 [9] metrics. This model is described in more detail in section *2.3.* and we will refer to it as **"W2B"** (please do not confuse this with the WikiBio dataset).

2. Overview of models and repositories

In this section, we provide an overview of the models that we used, we briefly describe their architecture and we discuss the contents, documentation and code in their original GitHub repositories. Moreover, in section *2.4.* we describe our approach, including the structure of our own repository, as well as the E2E metrics repository, i.e. the repository that contained the script we used to evaluate the models for the E2E dataset.

2.1. NTG - Wiseman et al. (2018)

As we briefly mentioned earlier, in Wiseman et al. (2018) [1], a **neural hidden semi-Markov model** (HSMM) decoder is proposed so as to alleviate the lack of interpretability and controllability in modern NLG systems which belong the encoder-decoder framework [6]. The main goal of the system is to extract and learn latent, template-like structures within the dataset which can later be used so that the generation follows a fixed plan for the content of the outputs, while also bringing the associations between a part of the generation and a record in the knowledge base (i.e. a specific entity) to the foreground.

In order to understand how the HSMM decoder can learn such template-like structures, we need to understand how a simple hidden Markov model works. A hidden Markov model (HMM) assumes that the hidden states of an unobservable process X can be learned by observing a dependent process Y. The emission probability is used in a single time step t. The only real difference of the HSMM is that its emission probabilities last multiple time steps.

The particular neural HSMM described in Wiseman et al. (2018)[1], combines the general HSMM structure outlined above along with the two main components that make modern NLG systems effective: **Long short term memory** (LSTM) recurrent neural networks (RNNs) [12] and attention mechanisms [13]. Thus, the HSMM takes into account the length of the input sequence,

Furthermore, the paper proposes two variants for their model that we experiment upon; the **autoregressive** and the **non-autoregressive**. The autoregressive variant of the HSMM makes use of an additional RNN which passes over all the preceding tokens that have been generated in each time step, so that the probability distribution of the next token to be generated depends on all previous generations. The non-autoregressive variant simply assumes that the token segments are independent and depend only with the associated hidden state of the HSMM and the input sequence.

As we will see in sections *4.1.3.*, after training either the autoregressive or the non-autoregressive variants of the NTG, two models are exported: **the decayed and the non-decayed versions of the model**. By default, an initial learning rate of 1.0 is used (with the Stochastic Gradient Descent optimizer) and in every epoch after the first epoch in which the validation log-likelihood fails to increase, the learning rate starts decaying by a factor of 0.5 per epoch [1]. If we explained the reason why the authors do this in layman's terms, we would say that when the loss does not decrease, the algorithm has probably found itself near a local minimum -although one would hope that it would find itself near the global minimum- of the optimization function and has to perform increasingly smaller steps in order to actually reach that specific point (thus, the name "gradient descent"). Thus, the *non-decayed model is the one for which the learning rate has not started to decay, while the decayed model is the final one which has -hopefully- converged*. In section *5.1.* we report the results for the autoregressive and non-autoregressive variants of both the decayed and the non-decayed models (4 different outputs).

The final detail which concerns the paper that we focused on, is the **qualitative evaluation** performed by the authors so as to actually demonstrate that the generations of the NTG are controllable and interpretable. The empirical analysis that they performed included calculating the average purity scores of the discrete states (i.e. templates) learned on the E2E and WikiBio datasets. The average purity is the percentage of a state's words that come from the most frequent record type that the state represents. Even though the authors claim to have added the necessary functionality to compute these purity scores [14; see 'closed issues' tab], we were not able to actually reproduce their results successfully (see section *4.1.3.*).

The source code of the original project is freely available on its own GitHub repository and is implemented in **Python 2.7**, combined with **PyTorch 0.3.1**, **CUDA 8** and the corresponding compatible versions of cuDNN [14]. This project uses both the E2E [3] and the WikiBio [5] datasets for the training and the generation, and the authors provide their own pre-processed datasets that fit the project's structure.

Within the repository, there are sample commands on how to reproduce the experiments, which are divided into and concern the following steps:

- **Training** for the autoregressive and non-autoregressive variants of the model for both datasets.
- **Viterbi segmentation** / template extraction for the two variants of the model for both datasets.
- **Text generation** for the autoregressive variants of the model for both datasets.

Apart from the pre-processed versions of the datasets, the authors also provide **four distinct pre-trained models** (the autoregressive and non-autoregressive variants for each of the two datasets), as well as ready-made **Viterbi segmentations with the extracted templates**. As we will see later on in section *4.1.*, it is unclear how the text files with the Viterbi segmentations are updated to include the text from the original datasets.

2.2. TGen - Dušek and Jurčíček (2016)

TGen, the model proposed by Dušek and Jurčíček (2016) [2], is described as a **sequence-to-sequence model** [15] **with attention** [13], as well as with added **beam search** and a **reranker** which penalizes outputs that are not close to the input meaning representations (MR). This model is the baseline of the E2E dataset. The sequence-to-sequence approach is an encoder-decoder RNN architecture which can work with sequences of tokens of variable length using minimal assumptions on the sequence structure [15]. Their work originally considered the BAGEL dataset [16] and was built for spoken dialogue systems. The inputs consist of dialogue acts (DA), i.e. representations of an action (e.g. inform, request, etc.) and the output could be of two different types, as the model had two modes of operation: producing either deep syntax trees (decisions on the syntactic shape of the output sequence) or natural language strings (combination of the sentence planning step and the text realization) [2].

In more detail, the sequence-to-sequence generator with attention used in their paper, encodes an input sequence into a sequence of encoder outputs and hidden states and then decodes the hidden states using a second LSTM-based RNN [12]. The beam search reads the sequence from left-to-right and keeps track of the log probabilities of the top n possible output sequences, while the reranker reranks the n-best beam search outputs and penalizes those outputs which differ from the original inputs [2].

The original project provided by the authors in its own GitHub repository, is written in **Python 3.6** and uses **TensorFlow 1.13** [17]. Also, this project, acting as a baseline for the E2E challenge, does not come with any specially crafted version of the E2E dataset, instead it uses the original one as it is provided. The authors provide neither their pre-processed dataset for training and generation, nor any pre-trained models. This means that every step must be reproduced in order to perform text generation for evaluating the performance. From the guidelines provided by the maintainers of the repository [17], the following steps must be reproduced:

- **Data pre-processing** and in particular, conversion into the format used by TGen
- **Training** on the converted training set of the E2E dataset.
- **Text generation** on the development set of the E2E dataset. Please note that we will hereby refer to the development set as the "validation set".
- **Data post-processing** and in particular, detokenizing the outputs, removing unnecessary whitespaces, etc.

2.3. W2B - Liu et al. (2016)

This project proposes a table-to-text (infobox-to-biography) generation model, i.e. a system which is able to generate a description of a textual table and in this particular case, a box containing biographical information from Wikipedia with field-value records (e.g. [Name, Chris Evans], [Occupation, Actor], etc.). The model follows the **sequence-to-sequence framework** [15] which consists of a **field-gating encoder** and a description generator with **dual attention mechanism** [4].

More specifically, the authors propose a structure-aware architecture which is basically an encoder-decoder [6] using LSTM [12] units with global and local addressing. The field-gating encoder combines field information and consent representation through the use of a field gate in the cell state of the encoder LSTM unit. The LSTM decoder generates text in a natural language using the structural representation of the paper, as well as a dual attention mechanism; field-level attention for global addressing and word-level attention for local addressing. Global addressing is used to decide which records of the input table should the model focus on, while local addressing is used to determine which particular words should the model focus on at each time step [4]. We should note that in Liu et al. (2018) [4], the authors conduct experiments with various variants of the model; the one that we have experimented upon and which we call "W2B" is the **"field gating Seq2seq with dual attention"**.

The code in the original GitHub repository is written in **Python 2.7** and uses **TensorFlow 1.0** along with the corresponding compatible versions of CUDA and cuDNN [18]. The authors provide their own stripped down version of the WikiBio dataset that fits the project's structure and has several lines removed, since these lines do not fit the authors' purposes, keeping only one biography per line.

However, the authors do not provide any pre-trained models and not even the actual library requirements for the reproduction of their experiments. Thus, from the guidelines provided by the maintainers of the repository [18], the following steps must be reproduced:

- **Data pre-processing** and in particular, extraction of the words, field types and position information from the original infoboxes of the WikiBio dataset.
- **Training** on the pre-processed training set of the WikiBio dataset, while also reporting the BLEU and ROUGE scores on the validation set for every epoch.
- **Text generation** on the test set of the WikiBio dataset as well as reporting the final BLEU and ROUGE scores.

2.4. Our project's repository

We have created a GitHub repository for our project, so that all the required software, scripts, files, models and data are easily accessible and can be fetched in a fast and reliable way, ensuring that anyone will be able to re-run the experiments and reproduce the results with ease. The only thing missing from our repository is a specially crafted Conda environment, but we have made sure that it gets set up in an automatic way, without any manual user intervention (as it is described in chapter 4).

The code in our GitHub repository is written using **Bash**, **Python** (using the respective Python version of each project we evaluate), **Docker** (version 19.03.12) and **Conda** (version 4.8.4). Docker is a piece of software that provides us with the capability of building an isolated environment (container) for our project, so that we do not mess with the host's OS settings, packages and software. The base image for our docker container is Ubuntu 18.04 with CUDA 10 support. We have also created a Python script that can be used as a text-based user interface which simplifies the complexity of using Bash and Python scripts from the command line with the proper formatting and naming of command-line arguments. There exist several directories within the repository which are listed and described below in alphabetical order:

- The 'docker' directory contains a Dockerfile, which is a script-like text file with instructions on building a custom docker image, as well as some instructions on how to set up and use Docker on a terminal in Ubuntu 20.04 LTS.
- The 'e2e-metrics' directory contains the Python and Bash scripts which can be used in order to automatically leverage the scripts and metrics provided in E2E NLG Challenge Evaluation[19] which help with evaluating the performance of any model that performs text generation using the E2E dataset [3], as well as some instructions on how to use them. It also contains a ZIP file with the output scores we measured.
- The 'ntg' directory contains the Python and Bash scripts which can be used in order to perform training, Viterbi segmentation, text generation and text generation post-processing, with the E2E dataset using the NTG software [14], written by Wiseman et al. (2018) [1], as well as some instructions on how to use them. It also contains several ZIP files with our own re-trained models, as well as the Viterbi segmentations and (post-processed) text generation our models performed.
- The 'tgen' directory contains the Bash script which can be used in order to perform training and text generation only with the E2E dataset, using the Tgen software [17], written by Dušek and Jurčíček (2016) [2], as well as some instructions on how to use them. It also contains some ZIP files with our own trained model as well as the (post-processed) generated text.
- The 'wiki2bio' directory contains the Python and Bash scripts which can be used in order to perform training and text generation with the WikiBio dataset using the W2B software[18], written by Liu et al. (2018) [3], as well as some instructions on how to use them. It also contains some ZIP files with our own trained model with the best BLEU score as well as the evaluation.

3. Datasets and Metrics

In this section, we give an overview of the two datasets used in our project; E2E and WikiBio. We also discuss the metrics that are used to evaluate each one of them.

3.1. E2E Challenge

The full E2E dataset contains contains **50,602 records** crowdsourced from a knowledge base in the **restaurant domain** in the form that we described in section *1.*, i.e. meaning representation-human reference ("MR"-ref) pairs (e.g. "name[Browns Cambridge], food[Fast food], area[riverside], familyFriendly[no], near[The Sorrento]" - In the riverside area near The Sorrento is the Browns Cambridge. It is a Fast food restaurant.). The dataset consists of 5,751 unique MRs, 50,602 human references (i.e. the number of the total records) and 945 distinct word types and thus, is bigger, richer and more complex than similar datasets [3].

The metrics used for the evaluation of the performance of the systems that took part in the E2E challenge are: **BLEU** [7], **NIST** [8], **ROUGE-L** [9], **CIDEr** [10] and **METEOR** [11]. The **scores** reported for **TGen** [2] and **NTG** [1] on the validation (referred to as "development set" in some of the respective papers) and test sets can be seen in Table 1 below.

Metric	TGen	NTG (non-AR)	NTG (AR)
Validation Set			
BLEU	69.25	64.53	67.07
NIST	8.48	7.66	7.98
METEOR	47.03	42.46	43.07
ROUGE_L	72.57	68.60	69.50
CIDEr	2.40	1.82	2.29
Test Set			
BLEU	65.93	55.17	59.80
NIST	8.59	7.14	7.56
METEOR	44.83	41.91	38.75
ROUGE_L	68.50	65.70	65.01
CIDEr	2.23	1.70	1.95

Table 1: TGen (baseline) and NTG (non-autoregressive and autoregressive variants) results on the validation and test set of the E2E dataset as reported in Novikova et al. (2017) [3] and in Wiseman et al. (2018) [1].

3.2. WikiBio

The WikiBio dataset contains **728,321** biographies from Wikipedia (Sep. 2015). It consists of all the biography articles that were listed in the WikiProject Biography which also have an infobox (a table). In Lebret et al. (2016) [5], the authors also specify that for each article, the infobox is encoded as a list of (field name, field value) pairs and the first paragraph is extracted, tokenized and lower-cased. Additionally, the WikiBio has been divided into three subsets; **training** (80%), **validation** (10%) and **test** sets (10%), while it should also be mentioned that, on average, the table is twice as long as the first sentence which acts as the human reference generation (53.1 vs. 26.1 mean tokens) [5].

Figure 1: An example from the WikiBio dataset [18], with a raw infobox from the database (left) for George Mikell and the corresponding first paragraph that is used as the reference generation (right).

In Lebret et al. (2016) [5], the models tested on the WikiBio dataset are evaluated using the **BLEU** [7], **ROUGE** [9] and **NIST** [8] metrics. The baseline that is used is an interpolated **Kneser-Ney** (KN) language model which has been implemented using the KenLM toolkit [19]. We actually downloaded, compiled and used the KenLM toolkit on an example text excerpt from the Bible. Nevertheless, we were not able to use it on the WikiBio dataset and produce any results, as *the documentation provided by its developer was insufficient*. Thus, the scores that we report below, in Table 2, concern only the NTG [1] and the W2B [3].

Metric	W2B	NTG (non-AR)	NTG (AR)
BLEU	43.65	34.2	34.8
NIST	-	7.94	7.59
ROUGE_L	40.32	35.9	38.6

Table 2: W2B and NTG (non-autoregressive and autoregressive variants) results on the test set of the WikiBio dataset as reported in Wiseman et al. (2018) [1].

4. Implementation Details and Improvements

In this chapter, we are going to present the implementation details for each of the 3 models that we tested (NTG, TGen and W2B), as well as the difficulties that we encountered and the modifications we made.

Below we list the 3 different types of hardware that we used to conduct our experiments and the project in general:

- 2 VMs, each one with 16 Intel Xeon E5-2650 (rel. in 2012) CPU cores, 32GB of system RAM and 500 GB of storage, running Ubuntu 20.04.
- Notebook PC with 16GB of system RAM and Nvidia GeForce RTX 2070 (Max-Q edition - rel. in 2019) with 16GB of VRAM, running Ubuntu 20.04.
- Desktop PC with 8GB of system RAM and Nvidia GeForce GTX 750 Ti (rel. in 2014) with 2GB of VRAM, running Arch Linux.

In each one of these systems, we have leveraged Docker. We have created a custom Dockerfile based on a pre-built CUDA container image, which contains all the necessary instructions for building an container image which contains all the necessary tools and data for creating a container through which we can execute all the commands and scripts required to achieve the desired results. In fact, the user interface (UI) we have created to simplify the use of our project from third-party users, runs the desired functions through the container.

Most of the tools and packages required for the projects are installed within the container through the Conda package manager, creating a virtual Conda environment for each

individual project with the required version of every package. The rest are installed either through 'pip' (the package manager for Python) in the respective virtual environment or through apt (Advanced Package Tool) system-wide (or container-wide). This makes the installation of different versions of the same software easier, avoiding conflicts between different versions of the Python programming language, PyTorch, TensorFlow, etc.. This makes our final project both easy to set up and portable. To ease up the setup and the execution, we have written a UI in Python 3 which serves as an abstraction layer which hides the bash commands required to perform any action through the container, from the end user. Through this UI, the user has a much better and more straightforward experience in experimenting with our project. This UI exists and should be executed outside the container, so that it can interact with the container.

4.1. NTG - Wiseman et al. (2018)

As we have already mentioned, we mostly focused on this project. The authors of this project provide us with the source code of their project, some pre-processed versions of the original datasets that are suitable for their model, four pre-trained models (one model for every dataset with two variants of these models, an autoregressive one and a non-autoregressive one) and the four corresponding Viterbi segmentations. So, anyone can use these provided tools and directly proceed to the text generation step, avoiding dataset pre-processing, model training and Viterbi segmentation (and template extraction).

The authors have performed all their training, Viterbi segmentation (and template extraction) and text generation leveraging a Nvidia GeForce GTX Titan X (rel. in 2015) GPU with 12GB of VRAM [14; see 'closed issues' tab].

We did perform the generation using the provided tools, but we also performed our own step-by-step experiments manually, all the way from training to the generation. We have created two Conda virtual environments for this project, one named "ntg_gpu" which is suitable for executing the code using the GPU and one named "ntg_cpu" which is suitable for executing the code using the CPU.

4.1.1. Documentation problems

There were several oversights and mistakes in the project's documentation and instructions, so we had to do quite a bit of research on why several errors appeared, as well as on how to adapt some of the given bash commands to our execution environment. Several issues have been resolved in the issue tracker of the GitHub project. However, the fixes to these issues have not been incorporated in the project's source code, so we had to go through the issue tracker and apply the fixes manually.

It should be noted though that *there are no clear instructions as to how those pre-processed datasets can be recreated from the original datasets* and that, as of 14/09/2020, it is an open issue left unanswered by the authors of the paper and the maintainer of the repository [14; see 'open issues' tab].

Furthermore, the 'wb_aligned' directory, as provided by the authors, only contains the pre-processed version of the WikiBio dataset. However, when we ran the software, trying to experiment using the WikiBio dataset, an error occurred, which turned out to be that the software required the original WikiBio dataset to be present in the 'wb_aligned' directory as well, having a specific naming scheme. This requirement was not mentioned in the documentation.

4.1.2. Implementation problems

As we mentioned above in section *2.1.*, this project is written in Python 2.7 with PyTorch 0.3.1. Unfortunately, PyTorch 0.3.1 is a very old version and only supports CUDA 8 which is unavailable from all Conda channels. So when we tried to install PyTorch 0.3.1 using Conda, we would always encounter package resolution failures which rendered the use of PyTorch 0.3.1 impossible. So, we had to use a more recent PyTorch version. In order to make our project compatible with the latest/current Nvidia RTX 2000 series GPUs, for which CUDA 10 or higher is recommended, we had to port the project's code to PyTorch 1.0, as this is the earliest PyTorch version that is compatible with CUDA 10. Thus, we first had to port the code from PyTorch 0.3.1 to 0.4.1 and afterwards, to 1.0. For each one of these portings, we had to read the corresponding release announcement with the method deprecations and replacements, before we proceeded to the actual porting process. Also, several trial-and-error execution attempts were required, so as to make sure that everything worked as expected, without any warnings and/or errors, since the porting process required the modification of several Python scripts from the original repository [14].

The software was unable to locate and read the dataset, an error that did not actually appear in the output in a clear and understandable way; the only error that we encountered in that case was an index error generated from iterating over an empty list. Because of that, it was difficult to pinpoint the reason why the execution resulted in an error. After performing manual code review and more trial-and-error attempts, we concluded that reason for the error above was that both in 'chsmm.py' and

'labeled_data.py' the the directory with the pre-processed and aligned WikiBio dataset is expected to contain the string "wiki" in its name, as demonstrated in the example execution instructions in the respective GitHub repository. However, the directory which contains this pre-processed and aligned WikiBio dataset, as it is provided by the authors, is named "wb_aligned". Editing the respective lines in both Python scripts so that the directory can contain either "wiki" or "wb" in its name fixed this error.

Moreover, there was a bad code structure regarding the command-line argument storing and examination, which resulted in the software always assuming the user requested for GPU offloading, even if they did not and even if no GPU was available on the system. Also, when a GPU was present, for some unknown reason, some of the data created during the execution of the PyTorch code were copied to the CPU, while other data were copied to the GPU. This resulted in errors when an operation was performed between data that were not stored in the same piece of hardware. This was a problem that was very difficult to resolve and which required several trial-and-error attempts; eventually we fixed the 'chsmm.py' (main execution file of NTG) by moving all the conflicting data from the GPU to the CPU, using the '.cpu()' function. Another very elusive and difficult problem in 'chsmm.py' that appeared after several hours of executing had to do with a specific loop in which a variable dynamically changed type in a very unexpected way, resulting in an execution error in some of the iterations since some instructions expected a specific variable type. The solution we implemented was to check the variable type before these instructions were executed.

4.1.3. Execution details and problems

For the execution we created four Bash scripts; each one with a specific purpose. The 'ntg_e2e_original.sh' and 'ntg_wb_original.sh' scripts are used to generate text using the pre-trained models and extracted templates that were provided by the authors, for each one of the respective datasets (E2E and WikiBio). The 'ntg_e2e.sh' and 'ntg_wb.sh' scripts are used to perform either training, or Viterbi segmentation (/template extraction), or text generation from the ground up, without any of the provided saved models or segmentations given by the authors. These scripts are responsible for activating the corresponding Conda virtual environment, depending on whether GPU execution was requested or not, as well as for overwriting the output files generated during previous executions. Part of these functionalities are provided through the UI.

We also used a Python script which was given by the authors of TGen [2] in its respective repository [17] to post-process the outputs (see section *2.2.*) of their model. In particular, we added functionality to remove the segmentations which are printed out in the text generated by the NTG by default and renamed it as 'postprocess_gens_e2e.py'. This script is located in the 'gens' directory and its output is stored in the 'gens_postprocessed' directory, in a format ready to be evaluated by the E2E NLG challenge evaluation metrics.

In addition to all the problems mentioned in the previous subsections, we faced a huge hardware limitation problem. Since the authors' GeForce GTX Titan X is a flagship grade desktop GPU and since our hardware is significantly less powerful, we could not perform the experiments as efficiently and quickly as the authors. In fact, some of the experiments could not be performed at all due to this limitation.

The authors of the original paper proposed a batch size parameter of 15 (command-line argument: -bsz 15) for the E2E dataset, as well as a batch size parameter of 5 (command-line argument: -bsz 5) for the WikiBio dataset, which are suitable for their hardware. However we found out that any GPU with 8GB of VRAM runs out of memory with these batch sizes for the respective dataset, as we will see later on (see section *4.1.3.2.*).

Finally, as we have already mentioned in section *2.1*, the authors of Wiseman et al. (2018) [1] claim that they have added the necessary functionality to compute the average purity scores for the discrete states that the model has learned; however there are no clear instructions as to which step does this computation concerns. When we actually tried it, we encountered an error which concerns the 'align_stuff()' function in 'chsmm.py' for which we believe that we have pinpointed the cause (see section *4.1.3.1.*).

4.1.3.1 E2E dataset

Using the GTX 750 Ti and the E2E dataset, the execution would result in a GPU out-of-memory error in most of the experiments, since the available memory is way too small; even after significantly reducing the batch size parameter. Reducing the batch size significantly (from 15 down to 1) allows the execution to proceed, avoiding the GPU out-of-memory error, but results in an increase of the number of batches in each epoch nonetheless. This in turn significantly increases the total time required to train the model. Therefore, the GTX 750 Ti was inadequate to successfully run all the experiments all the way to the end in an acceptable time frame.

We were able to successfully proceed through every step on the E2E dataset only through the use of the RTX 2070 GPU, after also reducing the batch size (from 15 down to 10). The

model training was completed successfully in approximately 18 hours (each variant took ~9 hours to train), the template extraction in 10 minutes and the text generation in ~4.5 hours (approximately 2 to 2.5 hours for each variant of the model).

Apropos of the CPU-only execution on the E2E dataset, the time required to perform any computation was orders of magnitude larger compared to using any GPU; even the GTX 750 Ti. Approximately one month was needed just to train the models on the E2E dataset. All in all, using only the CPU, we finally did manage to train all the variants of the models, extract the templates and generate the output texts, but only after several weeks of execution; a quite inefficient procedure.

Also, for some unknown reason, when we tried to extract Viterbi segmentations using our own trained models, the segmentation files only contain numbers (placeholder keywords) corresponding to the various extracted templates, but without any actual annotated tokens (e.g. "|144 |257 |112 |29 <eos>|208"), in comparison to the provided segmentations which contained both the numbers of the templates and the corresponding text (e.g. "The|168 <unk>|6 restaurant|164 is family friendly|64 .|78 <eos>|12"). This problem was also the reason as to why we were not able to perform the qualitative evaluation mentioned in Wiseman et al. [1], i.e. calculating the average purities of the HSMM's states.

4.1.3.2 WikiBio dataset

Unfortunately, after a lot of experimentation and trial-and-error attempts, we were not able to successfully re-train the NTG models on the WikiBio dataset, on any of the three available systems and thus, ended up using the pre-trained models that the authors provide. This is due to the fact that the training process of the WikiBio dataset requires more than 32GB of system RAM.

In more detail, we tried using the pre-trained models for the WikiBio dataset as well as the Viterbi segmentations given by the authors [14] in order to reproduce the last step of the experimentation, which is the text generation. Even after reducing the batch size significantly (from the proposed value of 5 all the way down to 1), the execution on our notebook system consumed all the 16GB of the available system RAM as well as more than 23GB of Swap space, while also using the RTX 2070 GPU. Notwithstanding, the process was finished successfully after ~9.5 hours, but the outputs were empty text files (containing only the values of the execution parameters given as command-line arguments). This is probably due to an error ('IndexError: too many indices for tensor of dimension 2') that appeared during the process. A similar problem emerged when we tried doing the same (using the models trained by us and the Viterbi segmentations provided by the authors) with the E2E dataset; again, the results turned out to be empty and an error appeared ('TypeError: "NoneType" object is not iterable'). After searching through the issues posted in the original GitHub repository, it seems that this is an error experienced by others as well and remains open as of 14/09/2020 [14; see 'open issues' tab]. Undeniably, the NTG required and still requires extensive debugging which was almost impossible for us to do due to the large execution times and with the project delivery deadline approaching, especially with the use of the WikiBio dataset.

To make matters worse, the 32GB of system RAM of the VM were not sufficient for executing any of the steps (model training, Viterbi segmentation and text generation) for the WikiBio dataset, as the execution would be always terminated by the OS's Out-Of-Memory (OOM) daemon for any batch size. Due to the shortage of system memory on the VM, even with an additional 32GB of swap space to perform page swapping, the VM was rendered totally unusable for this experiment since every software execution attempt, even just for text generation with the original authors' models and data, resulted in OOM error.

4.2. TGen - Dušek and Jurčíček (2016)

As we mentioned before, the authors of this project provide us only with the source code of their project since no other files such as pre-trained models were provided. However, quite fortunately, out of all these 3 projects, this was the only one with a complete and correct documentation, since nothing was missing and the instructions were the most straightforward with no errors at all. Thus, we had no trouble setting up the project, pre-processing the E2E dataset, training the model and generating the outputs, with **the total process requiring approximately 8 hours**.

However, unlike the other two competing projects, the authors have not tested it using a GPU, *so we are not sure whether it can be trained and tested using a GPU at all*. In fact, we experimented with training the model using a GPU and the corresponding GPU version of TensorFlow. From our experiments we notice no difference when executing using a GPU compared to when executing without one, which means that this software in its current form does not support GPU training and will not benefit from the existence of a GPU in the system, unless it is rewritten in a way that can leverage a GPU's capabilities.

We have created a Conda virtual environment for this project, named "tgen" and have kept only the directory

which is related to the E2E dataset; the rest of the files and directories have been removed.

4.2.1. *Implementation details*

This project is written in Python 3.6 with TensorFlow 1.13. We ported the software from TensorFlow 1.13 to TensorFlow 1.15, which is the latest backwards compatible version before the newer TensorFlow 2.0, which differs significantly and is backwards incompatible. The reason we decided to do this is that the more recent versions of PyTorch and TensorFlow from the default Conda channel (Anaconda channel) are built with support for modern CPU instructions like SSE2 and AVX, as well as Intel's Math Kernel Library for Deep Neural Networks (MKL-DNN). This means that some operations, when executed on the CPU, can be accelerated using specific CPU hardware (if present), a feature that we wanted to take advantage of. We also created a pull request upstream, which was accepted by the authors of the original project, so our changes are from now on included in the original TGen software.

In order to port TGen to TensorFlow 1.15 successfully, we had to read the release announcement with the method deprecations and replacements. After several trial-and-error execution attempts, we ensured that everything works as expected, without any warnings and/or errors (in theory, no errors could occur since there are no method removals between these two versions, just deprecations that prepare for the future TensorFlow 2.0). We also fixed some deprecation warnings regarding the YAML loader in Python, which are not related to Tensorflow at all.

The requirements for this project are provided inside the requirements.txt file, which can be parsed by 'pip' to install all of them automatically. However, since we prefer using conda, to have full control over the package versions and dependencies compatibility, we tried to install all of these through Conda in a virtual environment. Some of these requirements conflicted with Anaconda's version of "enum34". Also, the "RPyC" and "recordclass" libraries are unavailable from Anaconda. So, in the end, we installed these three requirements with the use of 'pip' along with KenLM and PyTreex which are in source code form, get fetched from their respective GitHub repositories and are built locally using the OS's compilers (gcc).

4.2.2. *Execution details and problems*

In a similar fashion with NTG (see section *4.1.3.*), we created a Bash script for the execution of the various steps of TGen, named 'tgen.sh'. This script can be used to perform either data pre-processing, model training, or text generation and post-processing from scratch. It is also responsible for activating the corresponding Conda virtual environment and for rewriting output files generated during previous executions. Part of these functionalities are provided to the user through the UI.

The first necessary step was to pre-process the E2E dataset in order to transform it into a form that can be used by TGen. Then, we had to train the model, using the pre-processed dataset as input. Finally, we had to generate the output texts, using the newly trained model, in addition to post-processing them so that they can be evaluated by the E2E NLG challenge evaluation script. The whole process was pretty straightforward.

4.3. **W2B - Liu et al. (2016)**

The authors of this project [18] provide us with the source code of their project and their own stripped down version of the WikiBio dataset, in which there exists one biography per line (in the original dataset, the biographies may take up multiple lines). Also, the authors suggest using a Nvidia GPU to run the software and have performed their experiments by leveraging a Nvidia GeForce GTX 1080 (2016) GPU with 8GB of RAM, with which the training required 36 to 48 hours to complete. We had to perform all the steps manually for our experiments, from the dataset preprocessing all the way down to the text generation. We have created two Conda virtual environments for this project, one named "w2b_gpu" which is suitable for executing the code using the GPU and one named "w2b_cpu" which is suitable for executing the code using the CPU.

4.3.1. *Documentation problems*

There were several oversights in the project's documentation and instructions and hence, we had to pinpoint the causes of several errors that appeared and additionally, modify the Bash commands for our execution environment. For example, the 'requirements.txt' file with the required packages for successfully executing the software was completely missing, so we did not know which packages were required for a successful execution. After going through the issue tracker on the project's GitHub repository [18; see 'open issues' tab] we concluded that apart from TensorFlow there was only one other missing requirement, the NLTK library. There was a code error in line 19 of the 'LstmUnit.py' script, for which the solution was posted in the issue tracker [18; see 'open issues' tab] but the fix has not been implemented into the software by the repository maintainers as of 14/09/20.

Moreover, the authors neglected to mention that the Perl script 'ROUGE-1.5.5.pl' needed to actually be made executable before the execution of the software, otherwise

the software was not able to compute and output the ROUGE metrics. This Perl script required some additional Perl modules to be installed, which were not mentioned in the documentation or the instructions and without which, the execution would result in an error. Finally, after several trial-and-error attempts and an excessive search, we found the names of the modules and how to install them in our docker container.

4.3.2. Implementation details

This project is written in Python 2.7 with TensorFlow 1.0. We ported the software from TensorFlow 1.0 to TensorFlow 1.15, which is the latest backwards compatible version before the newer TensorFlow 2.0, as we mentioned in section *4.2.1*. In that section, we had also explained the reasons as to why we did this, the advantages that it offers and the difficulties which emerge. Furthermore, TensorFlow versions from 1.13 and higher support CUDA 10, which is the lowest recommended CUDA version that is compatible with the latest/current Nvidia RTX 2000 series GPUs.

Since the dependencies we needed for the execution were installed in a Conda virtual environment, there was a problem in the communication between the system's Perl executable and the Perl's modules installed within the virtual environment. In order to overcome this, we installed both Perl and the required modules through Conda, pulling them from the Bioconda repository. Even then, the Perl script could not be executed because it would not use the Perl executable located in the virtual environment, so it did not load the needed modules. So, we gave up on this approach and decided to perform a system-wide Perl modules installation, so that the 'ROUGE-1.5.5.pl' Perl script which after some research was performed using the **cpanminus** tool. This way, everything was able to install and run seamlessly.

Furthermore, we had to change the way the required files and directories are created during execution, in order to avoid several errors. This is because in the original project, if several directories and files are not created before performing any model training or testing, the execution results in an error. It is also worth mentioning that no logging was performed when testing the model, since the 'write_log' function in 'Main.py' script was not being used during testing. The absence of logging when testing the model makes evaluating the model impossible. This issue has been mentioned in the issue tracker by another researcher and *we have offered a quite viable solution*, even though it remains open as of 14/09/20 [18; see 'open issues' tab].

Finally we removed the pointless copy of the project's Python scripts to the model saving directory, since it does not contribute anything to achieving the project's goal. We also introduced the capability of continuing the training of an existing model, if the training gets interrupted for some reason.

4.3.3. Execution details and problems

We have created a Bash script, named 'w2b.sh', which can be used to perform the data pre-processing, training, or testing steps of the project. This script provides the user with the option of using a GPU or not (by activating the corresponding Conda virtual environment), as well as the option of testing the model that has been pre-trained by us or training a new one (by rewriting the output files generated during previous executions). We have also created a Python script, named 'select_best_model.py' that automatically evaluates all the models that have been saved during training based on either their BLEU-4 [7] score or the F-measure of the ROUGE-4 [9] score on the validation set, depending on the user's preferences. Afterwards, the user can use the best model which has been selected and use it for testing, at which point they can decide whether they want the evaluated test metrics displayed and exported in a csv file or not. This last step is carried out using another Python script that we have created, named 'display_test_metrics.py'.

The first step required was to pre-process the WikiBio dataset in order to transform it into a form that can be used by W2B. Then, we had to train the model, using the pre-processed dataset as input. After the training was completed, we had to determine which is the best model (based on its BLEU score on the validation set) and use it to generate outputs on the test set, for which BLEU and ROUGE are evaluated.

The authors of the original paper proposed a batch size parameter of 32 (hardcoded in line 23 of 'Main.py') which is suitable for both their hardware and our RTX 2070. However we found out that any GPU with less than 8GB of VRAM runs out of memory with this batch size.

Furthermore, since our hardware is significantly less powerful than the authors', and due to the fact that the 2015 GeForce GTX 1080 was a high-end desktop GPU, we could not perform the experiments as efficiently and quickly as the authors, who claim that ~36-48 hours were needed to complete the training on their hardware.

Using the GTX 750 Ti, the execution would result in an out-of-memory error with the default batch size for the training, since the available memory is way too little. So, the GTX 750 Ti was inadequate to successfully run all the

experiments all the way to the end. Reducing the batch size from 32 down to 16, we were able to successfully complete the training. However, the training required more than 10 days to complete.

Using the RTX 2070 (Max-Q edition), **the training process was completed in approximately 123 hours** (~5 days), while more than a month was needed just to train the models using only a CPU. Every iteration during the training required ~6 hours using the CPU, while only ~1 hour was required, on average (to be exact the mean execution time for each epoch is 66.44 minutes), using the RTX 2070 GPU. We should also note that while the authors mention that the training stops after 50 epochs, the process continued until manually stopped by us after 112 epochs.

4.4. E2E NLG Challenge Evaluation metrics

The authors of this project provide us with the source code which is used to evaluate the natural language generation quality of a model that was trained on the E2E dataset and produced text generations on the validation set. We have created a Conda virtual environment for this project named 'e2e_metrics'. For the execution we created three Bash scripts, named 'e2e_metrics_ntg.sh', 'e2e_metrics_ntg_original.sh' and 'e2e_metrics_tgen.sh', each one for evaluating the corresponding model's generations (NTG - both ours and the original one - and TGen, respectively).

The aforementioned scripts activate the 'e2e_metrics' Conda virtual environment, evaluate the output generations (if the user does not wish to re-train the models, the outputs that we have generated are evaluated by default) and also provide the user with the option of displaying -as a table- and exporting the mean values of the metrics, as well as their standard deviations. By default, the BLEU, NIST, ROUGE, CIDEr and METEOR scores are evaluated for each line of the generations individually and exported into a tsv file which can be found within the 'output_scores/complete_scores' directory. The mean and standard deviation values for each metric are stored in the 'output_scores/mean_stdev_scores' directory as csv files.

It is worth mentioning that, in order to actually compute the evaluation metrics, we had to use the pre-processed E2E validation and test sets that we had created from TGen (see sections *2.2.* and *4.2.2.*) as the ground-truth references. Thus, we also had to modify the 'convert.py' (which can be found in the tgen/e2e-challenge/input directory) accordingly.

5. Experiments and Results

5.1. E2E Challenge

In this section, we report the results of the experiments that we conducted on the E2E dataset [3] for NTG [1] and TGen [2]. In particular, we report the mean and standard deviations of the BLEU, NIST, ROUGE-L, CIDEr and METEOR scores, as calculated for each line in the generation outputs by the corresponding script (see section *4.4.*).

First of all, in **Table 3**, we report the scores of the TGen on the validation and test sets of the E2E dataset, since it had the most straightforward documentation and execution.

Validation Set	
	Mean (St. Dev.)
BLEU	64.84 (17.99)
sentBLEU	66.68 (16.69)
NIST	5.44 (1.40)
METEOR	46.26 (11.69)
ROUGE_L	69.37 (12.84)
CIDEr	2.17 (1.03)

Test Set	
	Mean (St. Dev.)
BLEU	64.29 (19.04)
sentBLEU	66.14 (17.62)
NIST	5.46 (1.25)
METEOR	45.84 (10.67)
ROUGE_L	68.10 (13.44)
CIDEr	2.22 (1.05)

Table 3: Average metrics and their respective standard deviations for the results of the model used in Dušek and Jurčíček (2016) as evaluated on the validation and test set of the E2E challenge dataset.

As we can observe, TGen had a **similar performance on both the validation and test sets**; nonetheless, the mean value of all the metrics except CIDEr was higher when the model was tested on the validation set. However, the performance on the validation set *differs quite significantly* when compared with the original scores reported in Novikova et al. (2017) [3] and in Wiseman et al. (2018) [1] (see Table 1), something which is not true as regards the test set. For example, the original BLEU score reported for the validation set is **4.41 points higher** than the BLEU score of the model that we trained, while the same difference for the test set is only 1.64. The largest deviation, however, probably concerns the NIST score (8.59 originally vs. 5.46 reported here, for the test set), which is an adaptation of

BLEU that gives more importance to less frequent n-grams [8].

However, the most striking differences between the reported metrics and the ones that we have found, concern the NTG. Below, in **Table 4**, we list the evaluation metrics on the text generations which made use of the original trained NTG models and Viterbi segmentations/extracted templates provided by the authors of Wiseman et al. (2018) [1] (see sections *2.1.* and *4.1.*) for the autoregressive and non-autoregressive variants of the model. Although the scores for both the validation and the test set are reported in the original paper, the aforementioned tools that the authors provide, **concern solely the validation set** (there are no extracted templates for the test set).

	Non-Autoregressive	Autoregressive
BLEU	55.73 (20.22)	60.08 (20.31)
sentBLEU	58.43 (18.00)	62.31 (18.83)
NIST	4.19 (2.03)	4.83 (1.45)
METEOR	37.80 (8.93)	42.94 (15.05)
ROUGE_L	65.58 (11.76)	67.02 (14.61)
CIDEr	1.63 (0.93)	1.96 (1.08)

Table 4: Average metrics for the results of the original versions of the autoregressive and non-autoregressive variants of the model used in Wiseman et al. (2018) as evaluated on the validation set of the E2E challenge dataset, using the pre-trained models and Viterbi segmentations/extracted templates provided by the authors. The standard deviation of each metric is reported in parentheses.

The differences that can be observed between Table 4 and Table 2 are even larger, as the above scores (for the validation set) are closer to what Wiseman et al. (2018) [1] report for the test set. This is a discrepancy that should be further investigated; the BLEU score reported by the authors for the autoregressive variant on the validation set is approximately 7 points higher from what we have found.

In the two tables that follow (i.e. Table 5 and Table 6), we report the results for the NTG when we performed all the steps described in section *2.1.* **manually**. In particular, we list the scores of the **autoregressive** and **non-autoregressive** variants of the models (which also produce different Viterbi segmentations) and for each one of the two variants, the results generated by the **decayed** and **non-decayed** versions of the model (see section *2.1.*).

In **Table 5** below, we report the results from the **decayed model**. As we can see, the mean scores of all the metrics are significantly lower than everything we have seen so far. This is true for all metrics, except **NIST**, which is **higher** for the decayed model as compared to the one that we saw above (which used the tools provided by the authors) on the validation set, even by a small margin (4.35 vs 4.19 for the non-autoregressive variant and 5.07 vs. 4.83 for the autoregressive variant).

Validation Set

	Non-Autoregressive	Autoregressive
BLEU	48.98 (21.98)	53.87 (6.69)
sentBLEU	52.47 (18.75)	56.06 (18.31)
NIST	4.35 (1.96)	5.07 (1.02)
METEOR	37.38 (8.03)	40.29 (7.85)
ROUGE_L	57.79 (14.11)	61.43 (15.35)
CIDEr	1.21 (0.91)	1.45 (1.04)

Test Set

	Non-Autoregressive	Autoregressive
BLEU	41.11 (22.33)	47.59 (18.79)
sentBLEU	44.81 (19.46)	50.31 (16.82)
NIST	3.79 (2.14)	4.82 (1.13)
METEOR	35.37 (7.83)	38.36 (6.17)
ROUGE_L	53.47 (13.29)	58.29 (13.21)
CIDEr	1.18 (0.92)	1.37 (0.96)

Table 5: Average metrics for the results of the decayed versions of the autoregressive and non-autoregressive variants of the model used in Wiseman et al. (2018) as evaluated on the validation and test set of the E2E challenge dataset. The standard deviation of each metric is reported in parentheses.

Below, in **Table 6**, we report the results of the same experiments which were conducted for the **non-decayed** NTG model.

Validation Set

	Non-Autoregressive	Autoregressive
BLEU	37.91 (20.90)	44.21 (23.70)
sentBLEU	41.32 (18.69)	47.28 (21.94)
NIST	3.07 (2.08)	3.24 (2.38)
METEOR	29.63 (8.47)	30.93 (7.21)
ROUGE_L	48.21 (12.15)	51.51 (12.79)
CIDEr	0.83 (0.79)	0.90 (0.77)

Test Set

	Non-Autoregressive	Autoregressive
BLEU	33.02 (20.62)	37.63 (21.31)
sentBLEU	36.43 (18.68)	40.56 (20.30)
NIST	2.96 (2.24)	2.40 (2.15)
METEOR	27.67 (9.41)	30.02 (7.44)
ROUGE_L	45.24 (12.35)	50.25 (11.72)
CIDEr	0.79 (0.69)	0.79 (0.78)

Table 6: Average metrics for the results of the non-decayed versions of the autoregressive and non-autoregressive variants of the model

used in Wiseman et al. (2018) as evaluated on the validation and test set of the E2E challenge dataset. The standard deviation of each metric is reported in parentheses.

By comparing Table 5 with Table 6, i.e. the decayed with non-decayed versions of the model, one can observe that the decayed model is simply a lot better in all respects. Thus, it is evident that after the learning rate starts to decay, the model converges closer to the local minimum of the optimization function; it is also evident that this convergence is reflected in the scores of all the metrics.

To sum up, after conducting the experiments on the E2E dataset, the **TGen was found as the one that had the best performance overall**. However, *both the TGen and the NTG had some prominent differences between the scores that we report and the scores that were originally reported in their respective papers*. Due to the **insufficient documentation of the NTG** (as well as its lack of tools provided by the authors), we were not able to successfully complete its second step (i.e. the Viterbi segmentation / template extraction) and thus, were not able to reproduce and verify the scores that were originally reported in Wiseman et al. (2018) [1]. From all the metrics that were evaluated, *NIST appeared to be the least sensitive*, at least when comparing TGen (see Table 3), the NTG using the tools provided by the authors (see Table 4) and the decayed version of the NTG (see Table 5). However, it was also the metric that had the **most significant difference** with the ones reported in the original papers.

5.2. WikiBio

As we mentioned in section *4.1.3.2.*, we were not able to successfully conduct any experiments with the NTG on the WikiBio dataset and thus, cannot present any results at all.

The W2B model, however, was successfully trained after running for 123 hours on the RTX 2070 GPU. Thankfully, the authors have included functionality for exporting a log file with the validation scores for each epoch of the training process. In section *4.3.2.*, we discussed the problems that appeared with the Perl script which could be used to evaluate the ROUGE metric. As such, the original log file generated by the training process of W2B did not evaluate the ROUGE scores on the validation set for each epoch and the large execution times were prohibitive in re-training the model. Subsequently, after parsing and analyzing this log file, we could only choose the best model according to the BLEU scores. The best model is that of the **44th epoch** with a **BLEU score of 44.44** on the validation set. As we can see in Figure 2, the training process does not require more than approximately 60 epochs, as the BLEU scores on the validation set start dropping, even though the loss keeps dropping as well; this is a clear sign that the model is overfitting on the training data after that point.

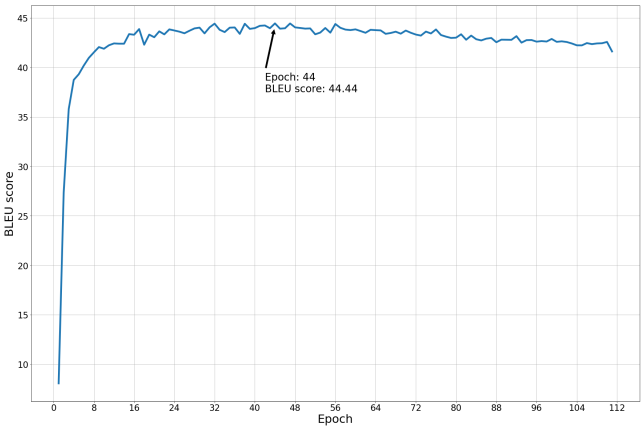

Figure 2: BLEU scores on the validation set of the WikiBio dataset for each epoch of the W2B model training process.

Below, in Table 7, we present the BLEU-4 and ROUGE-4 scores for W2B on the **test set** of the WikiBio dataset. The ROUGE-N scores evaluated by the code provided by the authors *are different* from the ROUGE-L scores that are used for the evaluation in the E2E dataset. In particular, here we report the **F-measure of the 4-gram co-occurrence statistics** [9]. Similarly, the BLEU-4 score here, is the 4-gram precision on blocks of text which has been found to *best correlate with monolingual human judgements* in Papineni et al. (2002) [7].

Metric	BLEU_4	ROUGE_4
Best model (by authors)	44.89	41.21
Best model (by us)	44.50	39.71

Table 7: BLEU-4 and ROUGE-4 (F-Measure) values for the best model (Field-gating Seq2seq with dual attention) in Liu et al. (2018) [4], as well for the model that we trained in this project, as evaluated on the test set of the WikiBio dataset.

We can observe that the results originally reported in Liu et al. (2018) [4] are very similar to ours, except for a difference of 1.5 in the value of ROUGE-4. In order to determine whether this difference is statistically significant, we should have performed the training and testing processes 5 or 10 times and report the mean and standard deviation of the metrics. Nevertheless, this would require a significant amount of time and would be inefficient. A different approach which could be followed and for which we provide the necessary functionality in the UI, would be to re-train the model just a single time and use the 'select_best_model.py' to report the scores of the two best models; the best in terms of BLEU-4 and the best in terms of

ROUGE-4 (note that it is not impossible that these two models actually coincide).

6. Conclusion

In this project we have conducted several experiments with recent models that are used in Natural Language Generation (NLG) from structured data. We focused on the E2E [3] and WikiBio [5] datasets, using the models described in Wiseman et al. (2018) [1], Dušek and Jurčíček (2016) [2] and Liu et al. (2018) [4]. Furthermore, we have created the necessary environments needed to use the models from end-to-end, a process in which we encountered numerous problems such as incompatibility and execution errors, missing or insufficient documentation, hardware limitations, actually implementing and evaluating the models, etc.; unfortunately, some issues remain unresolved. Finally, we put it all together in a Docker environment which ensures complete portability across different systems and we have created a user interface (UI), which provides the necessary functionality to reproduce and display our results or to conduct new experimentations, such as re-training and re-evaluating the models.

After attempting to reproduce the results that have been reported by the authors, we observed that W2B [4] had the most similar results with ours. On the other hand, TGen [2], the model used as the baseline for the E2E Challenge, was the one with the most coherent documentation and straightforward implementation. Finally, NTG [1], the model that we mostly focused on, was highly problematic in many respects and we did not manage to reproduce the results reported by its authors successfully. Thus, we were not able to assess the controllability and interpretability that the authors of the original paper guaranteed and that would provide us with a glimpse within the "black-box" structure of the modern NLG systems.

While it could be tempting to attribute this problem to the authors of the papers we worked with, its roots probably run even deeper. Academic and research institutions do not give incentives to researchers for creating and maintaining a decent GitHub repository, academic journal publishers and conference organizers do not have peer-reviewing processes for ensuring the quality and reproducibility of the reported results, while state and intergovernmental legislation does not provide any quality standards or guidelines.

It is our firm belief that research and innovation, especially as regards the fields of artificial intelligence and data science, could further be accelerated by simple and targeted measures which aim at the easy reproducibility of the models, code and results used in research papers.

ACKNOWLEDGMENTS

We are very grateful to GRNet, the National Infrastructures for Research and Technology, for providing us with the virtual machines (VMs) described in section *4.*, through ~okeanos-knossos, i.e. its cloud infrastructure as a service (IAAS). The VMs were of substantial help in our project.

REFERENCES

[1] Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2018. Learning neural templates for text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. October 31 - November 4, 2018, Brussels, Belgium. Association for Computational Linguistics, 3174-3187. https://doi.org/10.18653/v1/d18-1356

[2] Ondřej Dušek and Filip Jurčíček. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. August 7 - 12, 2016, Berlin, Germany. Association for Computational Linguistics, 45-51. https://doi.org/10.18653/v1/p16-2008

[3] Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E dataset: New challenges for end-to-end generation. In *Proceedings of the 18th Annual Meeting of the Special Interest Group on Discourse and Dialogue*. August 15 - 17, 2017, Saarbrücken, Germany. Association for Computational Linguistics, 201-206. arXiv: 1706.09254 Retrieved from https://arxiv.org/abs/1706.09254

[4] Tianyu Liu, Kexiang Wang, Lei Sha, Baobao Chang, and Zhifang Sui. 2018. Table-to-text generation by structure-aware seq2seq learning. arXiv: 1711.09724. Retrieved from https://arxiv.org/abs/1711.09724

[5] Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. November 1 - 5, 2016. Austin, Texas, USA. Association for Computational Linguistics, 1203-1213. https://doi.org/10.18653/v1/d16-1128.

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. October 25 - 29, 2014, Doha, Qatar. Association for Computational Linguistics, 1724-1734. https://doi.org/10.18653/v1/d18-1356

[7] Kishore Papineni, Salim Roukos, Todd Wards, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*. July, 2002, Philadelphia, USA. Association for Computational Linguistics, 311-318. https://doi.org/10.3115/1073083.1073135

[8] George Doddington. 2002. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*. March, 2002, San Diego, California. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 138-145. Retrieved from https://dl.acm.org/doi/10.5555/1289189.1289273

[9] Chin-Yew Lin. 2004. Looking for a few good metrics: ROUGE and its evaluation. In *Working Notes of NTCIR-4* June 2 - 4, 2004, Tokyo, Japan. National Institute of Informatics.

[10] Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. 2015. CIDEr: Consensus-based image description evaluation. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*. June, 2005, Ann Arbor, Michigan, USA. Association for Computational Linguistics, 4566-4575. https://doi.org/10.1109/cvpr.2015.7299087

[11] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: an automatic metric for MT evaluation with improved correlation with human judgements. In *11th Conference of the European Chapter of the Association for Computational Linguistics*. January 1, 2006, Trento, Italy. Association for Computational Linguistics, 65-72.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation.* 9, 8 (Nov. 1997), 1735-1780. DOI: https://doi.org/10.1162/neco.1997.9.8.1735

[13] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. arXiv: 1409.0473. Retrieved from https://arxiv.org/abs/1409.0473

[14] GitHub repository. harvardnlp/neural-template-gen. Retrieved from https://github.com/harvardnlp/neural-template-gen.

[15] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems 27 (NIPS 2014).* Curran Associates, Inc., 3104-3112. arXiv:1409.3215. Retrieved from https://arxiv.org/abs/1409.3215

[16] François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. 2010. Phrase-based statistical language generation using graphical models and active learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. July 11 - 16, 2010, Uppsala, Sweden. Association for Computational Linguistics, 1552–1561. Retrieved from https://dl.acm.org/doi/abs/10.5555/1858681.1858838

[17] GitHub repository. UFAL-DSG/tgen/. Retrieved from https://github.com/UFAL-DSG/tgen/.

[18] GitHub repository. tylipku/wiki2bio. Retrieved from https://github.com/tyliupku/wiki2bio.

[19] Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modifier Kneser-Ney language model estiamtion. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers).* August 4 - 9, 2013, Sofia, Bulgaria. Association for Computational Linguistics, 690–696. Retrieved from https://www.aclweb.org/anthology/P13-2121.pdf

[20] GitHub repository. tuetschek/e2e-metrics. Retrieved from https://github.com/tuetschek/e2e-metrics.