Universitat Politècnica de Catalunya (UPC)
BarcelonaTech

Facultat d'informàtica de Barcelona (FIB)

Bachelor's degree in Informatics Engineering
Computer Science Specialization

Final Degree Project

Exploring Reinforcement Learning in Natural Language Processing

---

Author:

Joan Francesc Muntaner González

Director:

Marta Ruiz Costa-Jussà
Computer Science (CS) Department

30 April 2021

Abstract

Natural Language Processing is the group of tasks that deal with "human" language. In this work we focus on machine translation, one of such tasks, that consists on automatically translating from one language to another. The main approach to this task is by means of deep learning techniques, particularly supervised learning. In this work we explore the application of reinforcement learning techniques to machine translation. We perform experiments using two extremely low-resource datasets (IWSLT14 German-English and FLoRes Nepali-English) as well as two different deep learning architectures: the Transformer, and a bi-LSTM network. We have found that common Reinforcement Learning techniques applied to the Transformer model require high amounts of GPU memory, and thus we have not been able to complete model training successfully. However, we have obtained positive results for the bi-LSTM experiments.

Resum

El Procesament del Llenguatge Natural és el conjunt de tasques que tracten amb el llenguatge "humà". En aquest treball ens centrem en la traducció automàtica, una d'aquestes tasques, que consisteix en traduir automàticament d'un idioma a un altre. La principal forma d'afrontar aquesta tasca és mitjançant l'ús de tècniques de aprenentatge profund, particularment d'aprenentatge supervisat. Aquest treball té com objectiu explorar l'aplicació de tècniques de aprenentatge per reforç a la traducció automàtica. Experimentem amb dos datasets d'extremadament baixos recursos (IWSLT14 alemany-anglès i FLoRes nepalès-anglès) així com amb dues arquitectures d'aprenentatge profund diferents: el Transformer i una red bi-LSTM. Hem observat que les tècniques habituals d'aprenentatge per reforç aplicades al Transformer requereixen molta memòria de la GPU i per tant no hem pogut completar exitosament l'entrenament dels models. No obstant, hem obtingut resultats positius als experiments amb bi-LSTM.

Resumen

El Procesamiento del Lenguaje Natural es el conjunto de tareas que tratan con el lenguaje "humano". En este trabajo nos centramos en la traducción automática, una de estas tareas, que consiste en traducir automáticamente de un idioma a otro. La principal forma de afrontar esta tarea es mediante el uso de técnicas de aprendizaje profundo, particularmente de aprendizaje supervisado. Este trabajo tiene como objetivo explorar la aplicación de técnicas de aprendizaje por refuerzo a la traducción automática. Experimentamos con dos datasets de extremadamente bajos recursos (IWSLT14 alemán-inglés y FLoRes nepalí-inglés) así como con dos arquitecturas de aprendizaje profundo distintas: el Transformer y una red bi-LSTM. Hemos observado que las técnicas habituales de aprendizaje por refuerzo aplicadas al Transformer requieren de mucha memoria de la GPU y por tanto no hemos podido completar exitosamente el entrenamiento de los modelos. No obstante, hemos obtenido resultados positivos para los experimentos con bi-LSTM.

Acknowledgments

I would like to thank Marta Ruiz Costa-Jussa for giving me the opportunity of doing a research project in a topic as interesting and challenging as neural machine translation. I would also like to thank all of my colleagues at the machine translation group from whom I have learned a lot.

This project's experiments have been carried out at the CALCULA TSC-UPC department GPU clusters, which made this work possible.

Finally, I would like to dedicate this thesis to my friends and family who supported me all the way through.

Contents

List of Figures

List of Tables

1 Context and Justification

1.1 Context

This is a Bachelor Thesis of the Informatics Engineering Degree, of the Computing Specialization, carried on at the Facultat d'Informatica de Barcelona of the Universitat Politecnica de Catalunya directed by Marta Ruiz Costa-Jussa, and within the Machine Translation research group [1] at UPC, part of TALP and IDEAI research centers.

1.1.1 Introduction

Natural Language Processing (NLP) consists of developing algorithms that allow computers to "understand" natural (human) language. It is a very broad topic that covers tasks such as spell checking, parsing or machine translation. In this work we will focus on the latter.

Current works on Machine Translation achieve good results between high-resource language pairs, that is, languages that have huge amounts of data available such as english-spanish translations; however, when faced with a low-resource language the system falters. Our goal is to apply reinforcement learning techniques in order to improve translations, specially in such cases.

Machine Learning Basics

Early approaches to Machine Translation used complex systems of formal grammars and rules, however current implementations make use of machine learning. Machine Learning is a sub- eld of Arti cial Intelligence that consists on devising an algorithm that learns a function from data itself instead of manually creating rules. There are di erent types of machine learning [2], namely *supervised learning*, where we use data tagged with its desired outcome; *unsupervised learning*, where we only have the entry data without its correct tag; and *reinforcement learning*, where an agent explores di erent actions to try to maximize reward within a certain environment [3].

In machine learning we want to learn a function that discriminates well between our data, but we have to mind a common problem, *over tting*. Over tting happens when our system learns random patterns from our data so that when encountered with new data it performs horribly. In order to avoid this we usually divide our dataset into two sets: the training set, and the test set [4]. We use the training set to learn our function and then evaluate our system using the test set.

Statistical Machine Translation

As we have said some of the rst approaches to MT were using complex

systems based on sets of rules. Quickly that developed into Statistical Machine Translation where we hoped to learn these rules from a large dataset (called corpus in NLP).

One important characteristic of machine translation that makes it a hard task compared to other ML tasks is that the function is neither one-to-one nor many-to-one as in many other applications of machine learning, but one-to-many: one sentence can be translated into di erent correct sentences. As such we have that the translation function that we want to learn is not deterministic and instead is modelled as a conditional probability $p(y|x)$ where we could have more than one target sentence with a high probability [5].

To build such models, the easiest approach is to use supervised training, that is to collect a data set of pairs of source sentences and translations. We denote by $D = (x^1; y^1); : : : ; (x^N; y^N)$ a data set of N pairs. We then want to score the model on how good it works on the data D. For that, we will use the log-likelihood which measures how good of a t our model achieves for that data.

$$L(; D) = \sum_{(x^N; y^N) 2 D} \log p(y^n | x^n;)$$

Our goal is to nd a con guration of the model (parameters) that maximizes this score L. This consists of nding a maximum likelihood estimator, a typical problem in machine learning. Now we are left with the challenge of modelling $p(y|x;)$. Statistical Machine Translation approach was a linear combination of many features and research focused on nding a good set of feature functions [7]. However, since 2013 neural networks have been the go-to for machine translation, namely neural machine translation (NMT).

1.1.2 Deep Learning and Neural Machine Translation

NMT makes use of the concept of neural networks, that is roughly based on human neurons [6]. The most basic concept of neural networks is a neuron itself. A neuron is a computation unit that takes n inputs and produces a single output. The parameters for each input value are known as *weights* and we also have a constant term called *bias*. The neuron takes each respective input with their respective weight, sums it all up with the bias and puts it into an activation function, usually a non-linear function like a sigmoid function. Figure 1 shows an example of such unit.

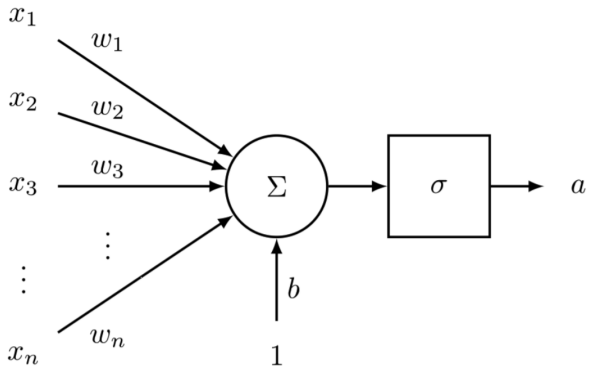

Figure 1: Diagram of a neuron. Source: [6].

By stacking several neurons, feeding the output of a layer of neurons as the input to another layer we build a system called feed-forward neural network that bene ts from all these non-linearities in each neuron to express complex patterns. The power from neural networks comes from all these patterns and the backpropagation algorithm [10], which permits adjusting thee speci c parameters that hinder the performance. Since our loss function is di erentiable, we can compute gradients at each step [6]. Making use of the derivation chain rule, we can update the correct parameters in order to reduce the loss of the system. Now we nd ourselves with an optimization problem, and there are multiple known approaches to it.

Almost all variants use some form of *gradient descend*: $^{(t+1)} = {}^{(t)} \quad r_{(t)} J$ where we update our parameters taking a little step in the direction of the gradient of the loss function at that step. One commonly used optimization to gradient descend is Stochastic Gradient Descend (SGD) where instead of calculating the entire gradient we take some samples to calculate the descend direction. In these algorithms, the rate of descend, in our previous equation, is called *learning rate* and it represents how big of a step we take in the direction of the gradient. We have to be wary of this parameter as we want to converge fast to a local minima but we want to avoid overshooting: when we skip through a local minima due to an overly big learning rate. Optimization over non-convex surfaces as is this case is a really complex problem and there are multiple techniques [6] involving dynamic learning rates such as *annealing* or exponential decay, where we start with a big learning rate that we reduce at each step. The optimizers with best performance currently make use of Adaptive Optimization Methods, where the learning rate is allowed to vary according to each parameter, among those are the Adam [11], AdaGrad or RMSProp optimizers [12]. Figure 2 shows a comparison of these optimizers when training a multilayer neural network over the MNIST image dataset.

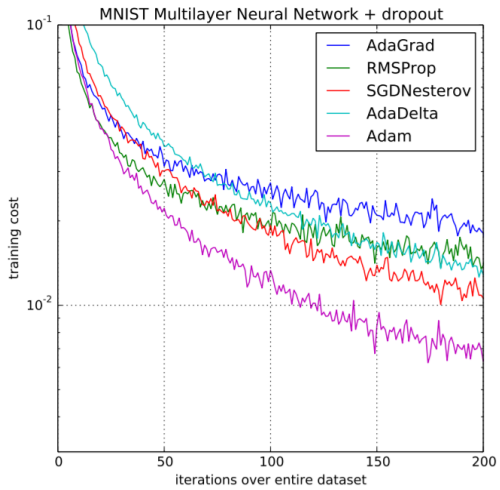

Figure 2: Comparison of di erent optimizers over multilayer neural networks on MNIST images. Source: [11].

As many machine learning systems, neural networks are prone to over tting and we could lose our ability to generalise data. In order to penalise over tting we can use some strategies [6] like regularization [8]. One of these techniques consists of adding a penalty term to the loss functions such that any big weights contribute negatively to the loss function. Another popular technique is *dropout* [9] where we "drop" some neurons randomly during training and reactivate them for testing.

It must be noted that training neural networks requires a high amount of computation power, on the other hand these computations can be performed in parallel as they do not usually have dependencies between them. Due to this highly parallel scheme, we are able to train neural networks using GPUs [5] that allow training with huge amounts of data in a feasible time.

We have already mentioned the feed-forwards network architecture, however with machine translation, we do not usually know the length of the input nor the output, in other words, we deal with variable-length inputs and outputs. To tackle this we make use of Recurrent Neural Networks (RNNs)[14]. The bene t of this model is that an RNN can maintain an internal state along an input, which makes variable-length inputs possibles [5].

It works on the idea that at step t we have a vector $h_{t\ 1}$ that represents the history of all preceding symbols, and with input x_t we calculate our new state h_t by using both the input and the previous state:

$$h_t = \quad (x_t; h_{t\ 1})$$

is an activation function that can consist of a single a ne transform with a non-linearity as with feed-forward. it is parametrized by .

$$h_t = tanh(Wx_t + Uh_{t-1} + b)$$

In this formulation [5] the parameters are the input weight matrix W, the recurrent weight matrix U and the bias term b. Moreover, several improvements to the vanilla RNN have been made based using a system of gates to capture long-term dependencies:

- Gated Recurrent Units (GRU) [16] have an Update Gate and a Reset Gate, the reset gate indicates to the system whether to keep information from a certain past state or not, conversely the update gate is responsible for how much of the new memory we pass to the next state, so it indicates if we update our state.
- Long Short Term Memory (LSTM) [13] is a more complex version of GRUs where we have a *forget gate*, that decides if we keep old memory, an *input gate* that decides if we take into account the new input, a *memory unit*, that maintains previous states information, and nally an *output gate* that controls what do we keep in the new memory cell.

Another possible implementation of RNNs is to make them bi-directional [6], where the system is able to "predict" future words by reading the corpus backwards. This model simply maintains two states for each direction, so it requires twice the memory space. In machine translation this can allow us to perceive data from both ways and not only use past context.

In using RNNs we must take into account the problems of Vanishing Gradients [15] and Exploding Gradients, that stem from the fact that as we apply the same weight matrices at each step the backpropagation algorithm takes an exponential form and can easily disappear or explode into huge numbers. To solve the exploding gradient we can use a simple heuristic to clip the gradient whenever it increases past a limit. For the vanishing gradient using a Recti ed Linear Unit (ReLU) [22] as an activation function instead of a sigmoid solves the issue.

The next step to work towards the state of the art transformer is the Encoder-Decoder architecture, namely Seq2Seq [17]. This model is an end-to-end model that uses two RNNs: an *encoder* that takes the model's input and encodes it into a xed-size context vector, and a *decoder* which takes this context vector and generates an output sequence. Usually these RNNs consist of several layered LSTMs units [6].

This Seq2Seq model poses the problem that gives the same importance to all the sentence for each output, as it compresses all the input information in one vector that is fed to the decoder. Here is when we introduce the concept of

attention mechanisms. The idea of attention is to provide the decoder with the ability to decide where into the source sentence it wants to focus [6]. The original Seq2Seq+Attention paper [18] proposal uses a bi-LSTM for the encoder with hidden states $(h_1; ...; h_n)$ and uses a context vector c_i that captures information of step i. Using the last step decoder hidden state s_{i-1} and hidden state h_j of the encoder we calculate a score $e_{i;j} = a(s_{i-1}; h_j)$ where a is a function with real values, for instance a single-layer Neural Network. We end up with values $e_{i;1}; ...; e_{i;n}$ that we normalise into vector $_i$ using a softmax function $_{i;j} = \frac{exp(e_{i;j})}{\sum_{k=1}^{n} exp(e_{i;k})}$. Finally we calculate our context vector by $c_i = \sum_{j=1}^{n} {}_{i;j} h_j$. This context vector intends to capture the relevant information of the input sequence at step i.

As translation is a complex subject due to how a sentence can have more than one correct translation, we need accurate measures to evaluate properly our systems. For this purpose we will use the Bilingual Evaluation Understudy (BLEU) metric [19]. This metric evaluates the fraction of n-grams (n consecutive words in a sentence) in the translation that also appear in the reference. We also include a brevity penalty to avoid common n-grams to achieve perfect scores, or else a translation like "the" would achieve a really high score as it is commonly present in all sentences. We calculate the BLEU score with the following equation, where p_n states the precision for n-grams, calculated with the aforementioned fraction (p_n = `#matched n-grams / #n-grams in candidate translation`).

$$BLEU = min(1; e^{(1 - \frac{len_{ref}}{len_{MT}})}) \prod_{n=1}^{1=2n} p_n^{1=2n}$$

A last topic we must address before tackling on the Transformer is how to encode words, in a sense, we cannot feed words to a neural network so we must find a good approach to encode sentences. A naive solution [5] would be to use one-hot encodings, where we have vectors of the size of the vocabulary and we have a 1 on the corresponding word and 0s elsewhere. However this representation is really problematic, firstly because we get huge vectors with sparsity problems and secondly because this encoding does not contain any information: all words are equidistant. Ideally we would want similar words close to each another.

For this precise purpose we use word embeddings, like Word2Vec [20]. These methods consist of learning a lower dimensional vector for each word with common dimensions around the hundreds (e.g. 100 or 300). There are a lot of algorithms to learn these vectors, some of which make use of different techniques like subwords, semantic relations or syntax. Learning these vectors from scratch can yield better results although it is very time consuming and either way there are a bunch of pre-trained word embeddings with similar performance.

1.1.3 The Transformer Model

The Transformer [21] stems from the need of parallelization to train systems with more data. To this point the main approaches used RNNs which are inherently sequential. The idea is to try to get rid of RNNs just keeping a more sophisticated attention. The particular attention used is called Scaled Dot-Product Attention, calculated using three inputs, queries and keys of dimension d_k and values of dimension d_v. In matrix notation we get:

$$\text{Attention}(Q; K; V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

This single attention only permits one way for words to interact with each other. To address this, they use multi-headed attention where they use 8 attention heads in parallel and concatenate its outputs. Both these attentions are illustrated on gure 3.

$$\text{MultiHead}(Q; K; V) = \text{Concat}(\text{head}_1; :::; \text{head}_8)W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q; KW_i^K; VW_i^V)$$

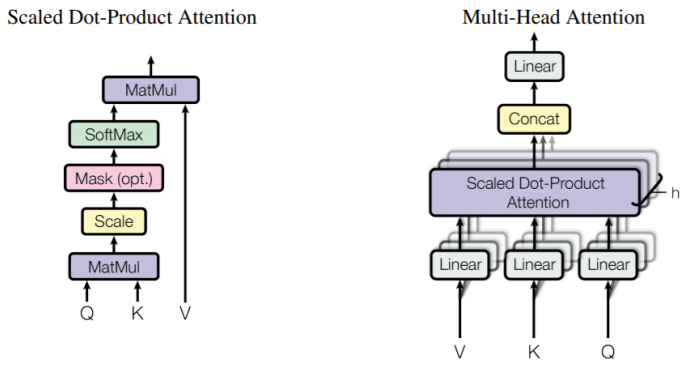

Figure 3: (left) Scaled Dot-Product Attention. (right) Multi-Headed Attention involves attention layers running in parallel. Source: [21]

Each block of the transformer has two sublayers: a multi-head attention mechanism and a position-wise fully connected 2-layer feed forward Neural Net with a ReLU [22] activation. Each of these also has a residual connection [24] (short-circuit) followed by layer normalization [23], which changes input to have mean 0 and variance 1. The encoder consists of 6 vertically stacked blocks as described. The decoder is also composed of a stack of 6 of these blocks with two slight additions: rstly we add a third sub-layer that consists of a multi-head attention over the output of the encoder stack; secondly we include a masking in the self-attention layer in order to prevent access to future positions of the

output. Both these blocks are shown in gure 4: an encoder block on the left of the gure and a decoder block on the right.

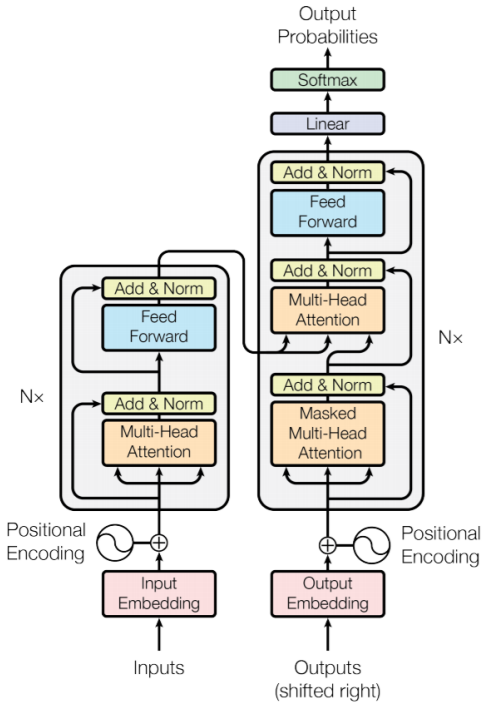

Figure 4: Transformer Model Architecture. Source: [21]

As we can see in gure 4 attention is used in three di erent ways:

- Encoder Self-Attention: all queries, keys and values come from the same place, the previous layer, allowing the encoder to attend to all positions in the previous layer.
- Decoder Self-Attention: similarly, with the addition of the aforementioned masking, we attend to all positions of the decoder up until that point.
- In "Encoder-Decoder" Attention layers we mimic the attention mechanisms described in Seq2Seq models [18]. We get information from all the positions of the input sequence, with the queries coming from the previous decoder layer an the keys and values come from the output of the encoder.

The Transformer model uses learned embeddings to transform input and output tokens into vectors we can feed into the model. At the end, after the last decoder layer, it passes the output through a linear transform (fully connected neural network) and the softmax function in order to obtain next-token probabilities.

The final feature we must go over is positional encoding: since the model does not contain any type of recursion or convolution we need some mechanism to inject information about the order of the sequence. Positional encodings are added to the embeddings in order to provide this information. There are many choices for positional encodings although the transformer uses a sinusoidal version ($PE_{(pos;2i)} = sin(pos/10000^{2i/d_{model}})$). Finally, to sum up, a complete picture of the model is shown in figure 4.

1.2 Stakeholders

Once contextualized, lets define the stakeholders of this project. Since this is a research project it does not really have a group of users we want to deploy our end product onto, because of this we will not be taking into account usability nor will we build an interface. The main beneficiaries of this work could be other MT researchers, particularly those inside MT UPC research group. If the goal of the project is achieved, and we find improvements via RL techniques, we could establish a baseline for future research into this topic. Furthermore, in case of not finding good results, it could serve as discouragement of exploring the techniques used for other research. Either way it will provide valuable insights.

In case it was a successful path, and this technique could be deployed into some company-owned translation systems, it could specially help those who need translation for low-resource languages, as we stated before.

Finally, the person in charge of this work could be benefited from this research and pursue more research on the topic in the future.

1.3 Justification and Previous Studies

As we have stated, in our project we will take as a baseline the vanilla transformer and we will try to improve results by applying reinforcement learning techniques. As such, we are taking an existing solution and making tweaks on it. That is because research in NMT is really hard and trying to come up with a new architecture of neural networks from scratch would be unfeasible with the resources available.

The RL path applied to NMT has not been explored much in current research, as it constitutes a fairly new approach. Reinforcement learning yields some of the best results in other deep learning applications such as AIs in videogames [37]. However, for NMT we face numerous challenges. Current RL-NMT systems only achieve improvements over already well-trained models, working best as a kind of fine-tuning for supervised learning models [25, 33].

1.3.1 Reinforcement Learning in NMT

In Reinforcement Learning we try to tackle problems by defining a *goal* and we expect the *agent*, the decision maker, to learn a *policy* (its behaviour) and choose *actions* according to that policy. The agent is supposed to learn this policy by interaction with the *environment* which in turn gives certain *rewards* to the agent [35].

Reinforcement Learning in NMT aims to optimize the evaluation metric (eg. BLEU) at training time [34]. By the previous concepts, we can say the NMT model is our *agent* that interacts with the *environment*, the previous words and the context vector. The parameters of the agent define a *policy* (the output probability), and the agent picks an *action*, a word from the vocabulary according to the probability distribution (policy) defined by the model (agent). Finally, once a complete sequence is generated, the agent is capable of observing its *reward*, the BLEU score. These definitions clearly translate the RL framework to the NMT case.

Recall that neural-network models (like the transformer) are trained with maximum log-likelihood estimation on parallel data, resulting in the following cross-entropy objective function:

$$L_{mle} = \sum_{(x;y) \in D} \log p_\theta(y|x)$$

The idea here is to try to introduce rewards like the BLEU score, denoted as $R(\hat{y}; y)$, where we compare the generated sentence $\hat{y}$ to the correct sentence y obtaining a certain score. With this function we aim to encourage the model to obtain high rewards, that is, high-scoring translations. For that purpose, the goal of the RL agent is to maximize the expected reward for all model outputs, defined as:

$$\begin{aligned} L_{rl} &= \sum_{i=1}^{N} E_{\hat{y} \sim p_\theta(\hat{y}|x^i)} R(\hat{y}; y^i) \\ &= \sum_{i=1}^{N} \sum_{\hat{y} \in Y} p_\theta(\hat{y}|x^i) R(\hat{y}; y^i) \end{aligned}$$

where $\hat{y}$ can be any of the possible candidate translation sentences (Y), which is exponentially large and makes maximizing this reward virtually impossible. For this purpose we use an approximation to the above expectation by policy gradient methods [38] like REINFORCE [36], used in [34], or Minimum Risk Training [26]. We opt for the REINFORCE approach.

1.3.2 RL training with Reinforce

Using REINFORCE, we approximate the expectation by sampling a single $\hat{y}$ from the policy $p(y|x)$, leading to maximizing the following equation:

$$\hat{L}_{rl} = \sum_{i=1}^{N} R(\hat{y}^i; y^i)\, \hat{y}^i \sim p(y|x^i), \forall i \in [N].$$

It is not easy to put RL in practice: one of the weaknesses of RL is that it is highly unstable and inefficient. There are however some methods to try to stabilize RL training [34].

Reward Computation

First of all it is important to define appropriate rewards for RL training. We need to consider two aspects, how to sample $\hat{y}$ and whether to use reward shaping. For sampling $\hat{y}$ we have two main approaches. The first one is *beam-search* [17] where we maintain a *beam* of top-K scoring candidates at each step, then for each of these sentences we append the K most-likely words, obtaining a set of $K \cdot K$ candidates, finally we select from this set the top-K translations with the largest probability. The other strategy is *multinomial sampling* [27] which selects each word by, as the name suggests, multinomial sampling over the output distribution. The choice between these two strategies reflects the classic RL *exploitation-exploration* dilemma [35] with multinomial paying more attention to exploring more diverse options while beam-search provides more accuracy by exploitation of the current model.

We can see in the previous equation that our RL system has the same reward for the entire sequence $\hat{y}$ even though the agent must take several actions before completing a whole sequence. Reward shaping [28] is a technique used to overcome this situation. In reward shaping we define an intermediate reward at each decoding step t denoted by $r_t(\hat{y}_t; y)$. In [29] they define $r_t(\hat{y}_t; y)$ as the increase of BLEU score with respect to y at step t: $r_t(\hat{y}_t; y) = R(\hat{y}_{1:::t}; y) - R(\hat{y}_{1:::t-1}; y)$, where $R(\hat{y}_{1:::t}; y)$ is defined as the BLEU score of $\hat{y}_{1:::t}$ with respect to y.

Variance Reduction of Gradient Estimation

The REINFORCE algorithm has a high variance in gradient estimation due to using a single sample $\hat{y}$ to calculate the expectation. In order to reduce this variance, in [30] it was proposed to subtract an average reward from the returned reward at each step t, then the actual reward used is $R(\hat{y}; y) - \hat{r}_t$ where $\hat{r}_t$ is the estimated average reward at step t, called *baseline reward* [31]. In order to calculate this baseline reward, one could sample multiple sentences and use its mean terminal reward.

Combining MLE and RL objectives

A simple but important strategy consists of combining the MLE and the RL losses. A simple way to do that is to linearly combine both objectives:

$$L_{com} = \quad L_{mle} + (1 \quad) \quad \hat{L}_{rl}$$

where is a hyperparameter that controls the trade-o and can be adjusted. The best tradeo s obtained in [34] are with = 0:3. However in [32] the best results are obtained with an between 0:02 and 0:1. This suggests that we will need to adjust the hyperparameter for our model.

1.3.3 Conclusions

To sum up, this work aims to explore the application of RL methods, particularly policy gradient methods like REINFORCE, to NMT. We will be using the vanilla transformer as our baseline and try to devise a system with better performance.

Summing up relevant related works, in [34] they explore REINFORCE policy gradient applied to the Transformer and di erent strategies and tricks to improve scores achieving an improvement of 0.75 BLEU score points (3.08%) in relation to the vanilla Transformer using RL without monolingual data. There is also a similar work [32] to ours which uses the same dataset (iwslt2014 de-en) and the same baseline (vanilla transformer). They mainly use the techniques described in [34] achieving an improvement of 0.83 BLEU score points over the baseline. Finally, in [25] they describe several weaknesses and challenges in current approaches of RL to NMT and propose strategies that could solve them. Furthermore, they study what and how these models learn in the context of NMT.

These models clearly de ne some weaknesses and challenges of the current RL approaches to NMT. Particularly in [25] they propose existing techniques in RL to address these challenges that have not been yet adopted in NLP. Our goal is to explore all these methods described in order to achieve or improve state of the art performance.

2 Project Scope

2.1 General Scope

The main objective of this project is whether we can improve the state of the art performance of the Transformer by implementing Reinforcement Learning techniques.

To accomplish such task we have de ned some objectives:

- Do research in Machine Translation state of the art.
- Do research in Reinforcement Learning techniques.
- Reproduce the state of the art results with the Transformer.
- Develop a new system applying RL techniques.
- Compare the proposals with the baseline objectively and rigorously by conducting the necessary experiments and draw conclusions out of the results obtained.
- Document all the work.

2.2 Scope Details and Requirements

The eld of MT is really large and advances quickly with research groups that have a lot of resources available. As such, although improvements to the Transformer architecture have already developed, we will be taking the vanilla transformer as the baseline for our project. It would be desirable to test our approach on di erent datasets, with di erent language pairs and to have extensive training and adjusting of hyperparameters. However, all this is beyond the scope of this project as it is too time-consuming and computation intensive. For that reason we will work on a small dataset with a single language pair.

As we have already stated before, this work in being a research project is not aimed at the market for end users, so we will not be taking into account usability and design. However, as any programming project, we aim to make the code readable and properly documented since it could be useful to other MT researchers.

2.3 Potential obstacles

There can be some obstacles or risks that appear during this project. Mainly:

- Deadlines. There are several deadlines we have to meet that may force us to reduce objectives if we lag behind planning. We can address this by following a good planning.
- COVID-19. COVID-19 is a disease that is greatly a ecting daily life in the whole world. In the scope of this project it could suppose sudden illness and a possible quarantine or inability to meet the tutor in person. However, as this project is programming-based it should not be much of an issue as meetings can be scheduled online and UPC resources are available through VPN.
- Individual project. As this work is in charge of only one person any setbacks they were to su er, could a ect the deadlines.

- APIs Documentation. In this work we make use of a variety of libraries and follow a series of papers that may not be extensively documented so we may have a hard time using them or reproducing results.
- Resources failures. This project is heavily dependant on a number of different resources, such as the personal computer, online services like Github, or the VPN to access UPC's GPU clusters. Any of those were to have a problem could cause an important delay or even loss of information or progress.
- Bugs. Errors in our code or some libraries could take long to detect or fix, or if going undetected, hinder our results.
- Insufficient computing power. We could find ourselves that the GPU clusters may have long queue times due to other researchers' computations or that our executions took too long (although the latter is not expected).
- Difficulty in applying RL techniques. We have already stated that previous studies hinted about the weaknesses of reinforcement learning for MT, and we are not experts in RL, so we could have a hard time applying RL to MT.

2.4 Methodology and rigour

Due to the tight schedule of the project the best approach seems to be using an agile methodology. However, as this project is individual, all of the teamwork does not apply here. The main approach will be to develop code and maintain constant feedback with the supervisor of the project (who could be seen as the client).

In order to maintain rigour in our experiments and make fair comparisons between different implementations it is important that we use the exact same training and testing set for both of them.

2.4.1 Tools

We will use a Github repository as a tool for version control and to facilitate sharing the code and preventing data loss as it is stored in Github's servers. We will use the different options Github provides such as different cards for tasks in progress, to-do or completed; or the different branches it provides.

In the developing matter, we will use Python3 since it is the most used Deep Learning programming language and we will use NLP libraries specific to Python. We will use PyTorch [39], an open-source deep learning library developed by Facebook AI, specifically the FairSeq module [40], which is particular to NLP and includes models we need such as the transformer. One of the main features of PyTorch is that it uses CUDA in its code, a NVIDIA-developed programming language to make computations on GPUs. This will ease all the calculations as

we will not have to worry much about the speci cs of the GPUs calculations as PyTorch gives it to us.

Finally, we have access to the aforementioned GPU clusters of the TSC department (CALCULA) so we can train our models. The dataset used in this work is the IWSLT 2014 German-English corpus [41].

3 Project Planning

The project o cially begins on 14th September, 2020, when the academic semester starts, the deadline to submit the thesis report is set for the 18th January 2021. It has not been decided a date for the oral defence yet, in case of any mishap it would be rescheduled to be defended in April, which would virtually double the amount of time available for the project. The project is then set to last 520 hours distributed in between the aforementioned dates, which comprises a period of about 4 months.

3.1 Resources

We will give an explicit enumeration of all the resources.

3.1.1 Human Resources

We nd three important human resources. Firstly, the author of this work, the researcher is the responsible for the development of the project, his tasks are to plan, develop, experiment, analyze and document the project. Next, the tutor is responsible for guiding the researcher in order to correctly achieve the goals of the project. Finally, the GEP Tutor helps the researcher complete all the project management tasks successfully.

3.1.2 Hardware Resources

The following resources will be used:

- Personal Laptop: Dell Inspiron 7577 i7-7700HQ 16GB RAM 256 SSD 1TB HDD. It will be used for programming and writing the documentation. it is the main work tool.
- CALCULA cluster: to train the models. The TALP research center has available 40 CPUs (Intel Xeon X5660), 8 NVIDIA GPUs, and 256 GB RAM.

Moreover we have to take into account resources to connect to the network (we need a router).

3.1.3 Software Resources

The following software resources are used, note that almost all of them are open source:

- Linux OSs and utilities, Linux Mint Cinnamon 19.2 installed in the personal laptop and Ubuntu 18.04 in the cluster.
- Atom: editor for coding purposes.
- Python 3, PyTorch and Fairseq: the aforementioned technologies we use, respectively, the programming language, the deep learning library and its sequence modeling (Seq2Seq) toolkit.
- Slurm 18.08, Git and Github: respectively, workload manager installed in the cluster, the control versioning software, and an online service to store Git repositories.
- Google Meet, Slack and Gmail: to maintain communication and meetings with the tutor.
- Overleaf and GanttProject: online tools to edit LaTeX and for building Gantt diagrams, respectively.
- German-English IWSLT 2014 corpus: dataset to carry on our experiments.

3.1.4 Researching Resources

In research projects is really important to attain knowledge from previous studies in the field of study as well as introducing one with the current methods. For that purpose we will need to read books, or papers. It is worth mentioning the main resource used to get introduced into Natural Language Processing is the CS224 Winter 2019 course of Stanford University[1].

3.2 Main Tasks

We identify the following main tasks. Since we use an agile methodology these could be modified. We will define duration, description and dependencies of all tasks. These will be summarised in table 1.

3.2.1 Thesis Management Course (GEP)

The project management task is one of the most important task group as it is the first we elaborate on. It defines all the context, scope, planning and budget. We identify the following subtasks in it:

- ICT Tools. We need to research the necessary software for several tasks like task planning or documenting.

[1] https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/

- Context and scope. We have to de ne the general objective of the project, contextualize it, de ne its scope and justify our approach.
- Project planning. In order to meet deadlines we devise a planning of all the tasks.
- Budget and sustainability analysis. It is important to assess the total cost of the project and the impact due to its development. We will make a budget and a sustainability report.
- Final project de nition. Taking into account the feedback provided by the GEP Tutor, we have to compile the previous three tasks into a nal document that de nes the initial assessment of the thesis.
- Meetings. Online meetings due to covid are scheduled on demand but the expectation is to do them biweekly. We will discuss the current status of the project and the following steps. We expect to increase the frequency of meetings closer to the end of the project.

3.2.2 Research/Learning

This being a research project involves a huge part of researching and learning. We identify two subtasks in here.

- Natural Language Processing with Deep Learning speci cs. We familiarise ourselves with the current approaches to NLP and particularly machine translation.
- Reinforcement Learning techniques. We need to learn about RL methods and how to apply them to the current MT methods.

3.2.3 Initial development

The developer will familiarise himself with the cluster installing the required dependencies. We also will run the baseline and try to reproduce the results, we have allotted extra time in case it does not achieve right away the score we expect and need to make any tweaks. At the end of this phase we should have laid down the bases to start developing the new RL approach.

3.2.4 New Models Development

This task will involve the development of new models with RL approaches. This is a really exible task as we might develop more than one new model. The design of the models will evolve with the feedback provided by the tutor.

3.2.5 Experiments

Since the testing and experimenting of our models can take a huge amount of time, we expect to have our MT system fully coded before starting tests. As stated, there is the possibility of developing more than one model but in that case we expect to have cycled fully through this task for that speci c model.

3.2.6 Documentation and defence

Although the documentation of the project will be updated constantly simultaneously with all the other tasks, at the nal stage of the project the thesis document will be the main focus. Finally, at the very end we will need to prepare the oral defence of the project.

ID	Name	Time (h)	Dependencies	Resources
T1	Project management	95		
T1.1	ICT Tools	3		PC, R
T1.2	Context and scope	25	T2*	PC, overleaf, GT, T, R
T1.3	Project planning	10		PC, overleaf, Ganttproject, GT, R
T1.4	Budget and sustainability analysis	15	T1.3	PC, overleaf, GT, R
T1.5	Final project de nition	25	T1.3, T1.4, T1.5	PC, overleaf, GT, R
T1.6	Meetings	17		PC, T, R
T2	Research	60		
T2.1	NLP research	25		PC, papers, books, stanford course, R
T2.2	Deep reinforcement learning	35	T2.1	PC, papers, books, R
T3	Initial development	70		
T3.1	Clusters/software con guration and installation	10		PC, clusters, T, R
T3.2	Baseline reproduction	25	T2.1	PC, git, programming language, clusters, R
T3.3	Preliminary coding and design with RL	35	T2	PC, git, programming language, clusters, T, R
T4	New models development	115		
T4.1	Programming	65	T2, T3.3	PC, git, programming language, clusters, T, R
T4.2	Debugging and testing	50	T4.1	PC, git, programming language, clusters, R
T5	Experiments and analysis	100		
T5.1	Experiments design and preparation	5	T3, T4	PC, R
T5.2	Experiments conduction	55	T3, T4, T5.1	PC, datasets, clusters, R
T5.3	Analyze and draw conclusions	40	T5.2	PC, results, R
T6	Documentation	80		
T6.1	Collect all the information obtained	10	T5.3	PC, results, R
T6.2	Writing the documentation	50	T1,T6.1	PC, results, overleaf, R
T6.3	Prepare oral defence	20	T6.2	PC, results, overleaf, project resources, R
Total		520		

Table 1: Summary of the tasks de ned in the project.
: The task that originates the dependency does not need to be complete in order to start the task.
GT: GEP Tutor
T: Tutor
R: Researcher

3.3 Risk Management: alternative plans

Although our planning is feasible, we should take into account the possible delays from the obstacles and potential risks detailed in section 2.3 (potential obstacles). In this section we introduce solutions in form of new tasks and rescheduling.

- Personal problems: in case of serious personal problems like illness the project would be rescheduled to be presented in April, which would give an extra 4 months of work.

- Scheduling: If we were behind schedule due to a bad estimation of tasks duration we could increase the working hours or use the weekends and holidays. The number of working hours per week could rise from 30h/week to about 80h/week. In case of a severe setback in scheduling we could as well reschedule the project for April.
- Resource failure: All the work will be backed up constantly in Github and the cluster, this redundancy should prevent any data loss. In case of a failure in the developing laptop we would resort to using the university PCs if COVID-19 allows for it or look for buying a new one.
- Bugs: Constant cycles of testing should avoid bugs, however it is frequent for a bug to go overlooked. In the case of a critical bug or a bug discovered too late into the project we would have to resort to rescheduling to April. We anticipate an increase of about 30 hours in case of minor bugs that delay us.
- APIs Documentation: In case of an API, library or research paper was lacking documentation or had any problem we could open up an issue in Github to ask information. In case of extreme difficulty we could consider using some other library. We anticipate 30 hours as well from the aforementioned extra hours allotted for unexpected events.
- Topic-specific problems
  - Insufficient computing power: if the computing power of the clusters is not enough for us, we could consider reducing the training set into a subset to train faster. We could also reduce the number of experiments.
  - Difficulty in applying RL techniques: although papers apply RL techniques to MT, they are not very successful, in case of not finding better results, we could try alternatively using them with RNNs or another simpler architecture.

  We also anticipate around 30 hours in case of any of these problems arose.

In total we have anticipated 90 hours of deviation from our original plan. In case of having to reschedule to April we would have approximately 520 extra hours, and we even could have more dedication to the project as we would not have any other enrolled courses.

3.4 Gantt Chart

Figure 5 details in a Gantt chart how we expect to distribute the project tasks.

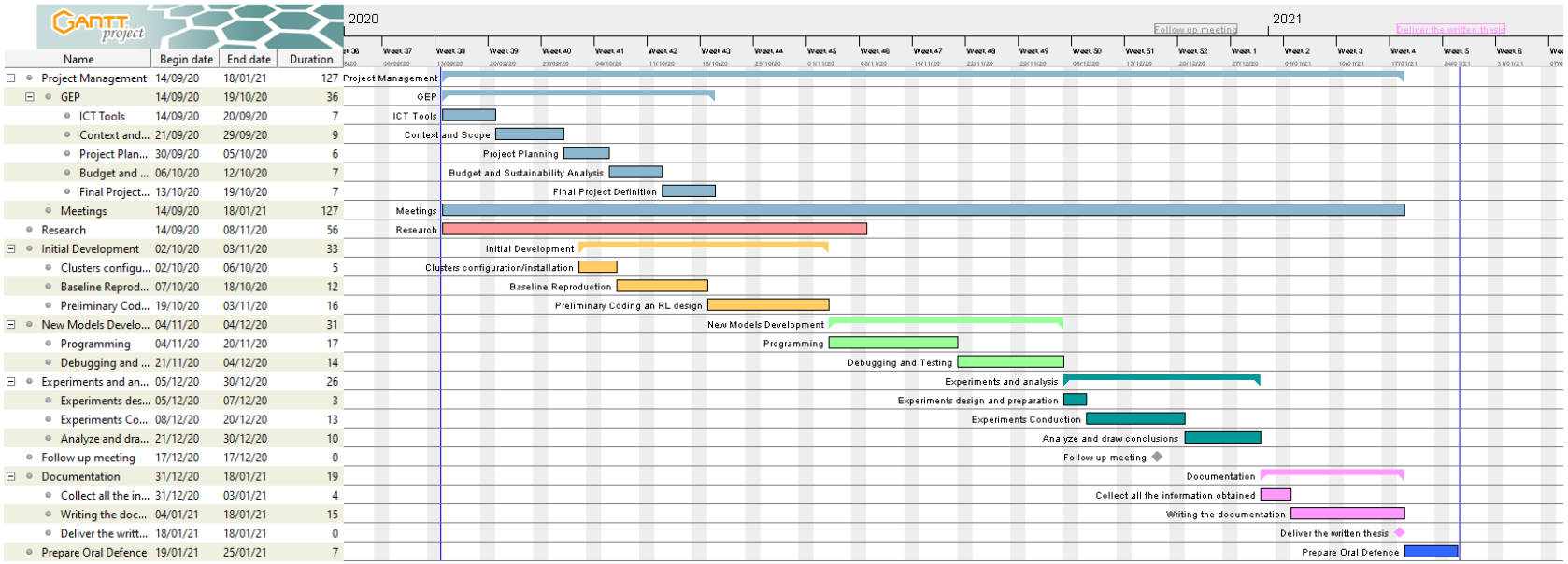

Figure 5: Gantt chart of the tasks de ned in the project

4 Budget

4.1 Identification of costs

We identify the costs of the project by the CPA (Personnel Cost by Activity), GC (General Costs), Contingency Costs and Incidental Costs.

4.1.1 Personnel Cost by Activity (CPA)

In order to calculate the CPA we need to de ne di erent roles for each task in the Gantt diagram. By doing this we will be able to estimate the cost of each task. We de ne the following roles:

- Project Manager: responsible for planning and guiding the correct development of the project.
- Software Developer: responsible for coding and testing the models.
- Deep Learning Researcher: responsible for designing and analysing the models as well as drawing out the pertinent conclusions.

Table 2 details the pay estimation for each of these roles.

Role	Net Income (e)	Gross Income (including SS) (e)	Hourly Wage (e)
Project Manager	46.492	62.764	35.86
Software Developer	30.370	41.000	23.43
Data Scientist	37.321	50.383	28.79

Table 2: Annual pay of the di erent roles de ned for the project. Salaries estimates extracted from Glassdoor[3] using Barcelona as location. We have assumed 1750 yearly work hours in order to calculate hourly wages.

Given the estimation of the income per hour of each role we can de ne the Personnel Cost per Activity for each task in our project. Table 3 details extensively the CPA per task where we can see that the total CPA amounts to 16001,59e.

4.1.2 Generic Costs (GC)

Among the generic costs we nd the housing, electricity, and hardware resources such as the laptop and the cluster. We also use software resources albeit they are all free of cost so we do not explicitly list them here.

- Cluster: we have estimated 240h of cluster usage for this project. As clusters are not really o ered for purchase we will estimate this cost by looking at renting prices of GPU clusters[4]. As such we estimate this cost to amount to 100.8e.

[3] https://www.glassdoor.es

[4] Although we can not nd the exact con guration we have in the server, we have taken the price of a Google K80 GPU (https://cloud.google.com/compute/gpus-pricing) for the EU-west1 servers (Belgium). This amounts to 0.49$/h, the equivalent of 0.42e/h at the moment of writing this.

ID	Name	Total Hours	Dedication (hours)			Cost (e)
			Project Manager	Software Developer	DL Researcher	
T1	Project management	95	95	17	17	4294,44
T1.1	ICT Tools	3	3	0	0	107,58
T1.2	Context and scope	25	25	0	0	948,72
T1.3	Project planning	10	10	0	0	463,04
T1.4	Budget and sustainability analysis	15	15	0	0	694,56
T1.5	Final project de nition	25	25	0	0	1105,38
T1.6	Meetings	17	17	17	17	870,72
T2	Research	60	0	0	60	1727,4
T2.1	NLP research	25	0	0	25	719,75
T2.2	Deep reinforcement learning	35	0	0	35	1007,65
T3	Initial development	70	0	30	40	1854,5
T3.1	Clusters/software con guration and installation	10	0	10	0	234,3
T3.2	Baseline reproduction	25	0	15	10	639,35
T3.3	Preliminary coding and design with RL	35	0	5	30	980,85
T4	New models development	115	0	95	20	2801,65
T4.1	Programming	65	0	45	20	1630,15
T4.2	Debugging and testing	50	0	50	0	1171,5
T5	Experiments and analysis	100	0	0	100	2879
T5.1	Experiments design and preparation	5	0	0	5	143,95
T5.2	Experiments conduction	55	0	0	55	1583,45
T5.3	Analyze and draw conclusions	40	0	0	40	1151,6
T6	Documentation	80	20	0	60	2444,6
T6.1	Collect all the information obtained	10	0	0	10	287,9
T6.2	Writing the documentation	50	0	0	50	1439,5
T6.3	Prepare oral defence	20	20	0	0	717,2
Total		520	115	142	297	16001,59

Table 3: Personnel costs by activity. Note that the meetings concept accounts for every role as we suppose all of them attend the meetings.

- Space and Internet: mainly due to SARS-CoV2 and government recommendations to preferably work at home, we can consider home to be the designated work space. Monthly rent amounts to 420e (shared house amongst three people) including services (water and electricity). Of those, 11e/month (33e divided amongst 3 people) go to the Internet connection. The rent (per person) excluding services amounts then for 367e/month. As the project is carried on in the span of ve months (September to January both included) we get a total of 420e 5 = 2100e that go to rent. Speci cally 367e 5 = 1835e amount to the house itself, 11e 5 = 55e are due to Internet service and the rest 2100e 1835e 55e = 210e are on account of services (electricity, water and gas).
- Laptop: we have to calculate the amortization of the laptop. Its purchase cost was 1129e, we estimate a lifespan of 4 years and 1750 yearly hours (as previously mentioned in Table 2) of usage. Given that we use it for 520h in this project, we get an amortization cost of $\frac{1129e}{4 \text{ years} \quad 1750h/year}$ 520h = 83.87e.

Table 4 summarizes all these Generic Costs that amount to a total of 2284.67e.

4.1.3 Contingency Cost

In order to give room for unexpected expenses due to unplanned events we consider a contingency fund. For this purpose, we add an extra 15% of the previously calculated costs (CPA+GC) as a contingency fund. Hence, contingency fund = (CPA + GC) 0:15 = (16001:59 + 2284:67) 0:15 = 2742.94e.

Activity	Cost (e)
Cluster	100.8
Space	2100
Housing	1835
Internet	55
Services (electricity, water, gas)	210
Laptop amortization	83.87
Total	2284.67

Table 4: Summary of Generic Costs.

4.1.4 Incidental Costs

We also have to take into account the cost of having to deviate into the alternative plans described in section 3.3 (risk management).

- Minor Bugs: we would have to add the 30 extra planned hours for personnel distributed between software developer and data scientist. This would represent an increase of 783.3e. We also would have to take into account the amortization of the laptop, that would add 4.84e. Finally we would have to calculate 30 hours of additional cluster usage that would amount to 12.6e totalling 800.74e for this incident.
- APIs Documentation: similarly to the minor bugs but excluding extra cluster usage, this incident would amount to 788.14e.
- Topic-speci c problems: for this incident we would only consider the extra 30 hours for the data scientist (amounting to 863.7e) and the laptop amortization (4.84e) giving us a total of 868.54e.
- Re-scheduling to next term: in case of having to reschedule to April, we would double the working hours, and as a result the personnel costs and laptop amortization costs would also double. We would also double the computing hours on the cluster as well, doubling its cost. Finally, we would have to consider three extra months of space and internet costs (February to April included). All of these would amount to an extra 17446.26e.

All these incidental costs are summarized in Table 5 and taking account their probability of happening we calculate a total risk exposure of 2598.57e.

4.2 Total Cost

The total cost of the project is summarized in table 6.

Incident	Probability	Estimated cost	Risk Exposure
Minor Bugs	50%	800.74e	400.37e
APIs Documentation	30%	788.14e	236.44e
Topic-Speci c Problems	25%	868.54e	217.13e
Rescheduling to next term	10%	17446.26e	1744.63e
Total	-	-	2598.57e

Table 5: Summary of Incidental Costs.

Activity	Cost
Personnel Cost by Activity (CPA)	16001,59e
Generic Costs (GC)	2284.67e
Contingency Costs	2742.94e
Incidental Costs	2598.57e
Total Cost	23627.77e

Table 6: Total costs.

4.3 Budget Control

It is very possible that same as with the time planning, budget estimations will not be exact. In order to tackle this, we introduce budget control mechanisms that allow us to manage budget deviations. Whenever we nish a task we have to calculate the deviation in all costs where the task is involved. In order to do so we will use the following calculations:

- Human resources deviation: it appears when the cost of the personnel or the real hours worked di ers from the estimated. It is calculated as follows for each role, and the total is the sum of every role deviation.

$$\text{HR Deviation} = (\text{Estimated cost per hour} - \text{Real cost per hour}) \cdot \text{Real Hours}$$

- Cluster usage deviation: we might not end up using the cluster all of the hours we expect or on the contrary need to use it more. We can calculate this deviation with the following equation.

$$\text{Cluster Usage Deviation} = (\text{Estimated usage hours} - \text{Real usage hours}) \cdot 0.42\text{e (price per hour)}$$

- Amortization deviation: if the laptop usage di ers from the expected its amortization cost will vary as well. We can calculate this deviation

with the following equation.

$$\text{Amortization Deviation} = (\text{Estimated usage hours} - \text{Real usage hours}) \cdot \frac{1129\text{e}}{4 \text{ years} \cdot 1750\text{h/year}} \cdot (\text{amortization cost per hour})$$

- Total cost deviation: amounts for the deviations in tasks costs, without taking into account contingencies or incidents.

  $$\text{Total Cost Deviation} = \text{Estimated general costs} - \text{Real general costs}$$

  By doing this we can monitor the budget and control exactly the cause and amount of deviation we get at each point. In case the total deviation was negative we would have to use the contingency funds.

  The last step would be to control incidents. In the event of an incident we would use the budget allotted for that purpose. Moreover, at the end of the project, we will be able to calculate an incidental costs deviation by subtracting the real incidental costs from the estimated incidental costs.

5 Sustainability Report

We will analyze the sustainability of the project in terms of the sustainability matrix, which has three dimensions. Project put into production (PPP), exploitation and risks. It must be noted that each cell is evaluated with a number from 0 to 10 where higher is better (in the risks dimension a higher number implies a lower risk).

	PPP	Exploitation	Risks
Environmental	Consumption of the design: 9	Ecological footprint: 9	Environmental: 9
Economic	Invoice: 7	Viability plan: 7	Economic: 6
Social	Personal impact: 8	Social impact: 7	Social: 5

Table 7: Sustainability matrix.

5.1 Economic Dimension

The budget of the project with its respective cost estimates can be found on previous section. We have obtained the average salary from job search websites, but it must be noted that this salary is an average from annual full-time contracts. In our case we do not o er as such good conditions and it might be the case that nding workers for a part-time job of a single project is harder, and as such candidates demand higher payrolls in order to be hired. Likewise, the social security cost calculation has been estimated as a 35% increase, but

in reality this value may not be accurate. Finally, (due in part to SARS-CoV2) we have assumed that this project is carried on at home, whereas in a real life situation where the virus was not a ecting daily life, this could be extrapolated to renting a co-working space or an o ce. On the other hand, apart from these possible issues, the project seems economically viable and the budget seems pretty reasonable.

The state of the art in NLP research is mainly carried on at North-American universities and huge organisations like Google, Facebook, OpenAI. They have available really powerful resources that by economy of scale should be cheaper per iteration than what we have here at Barcelona. However, looking at this project's budget, it is clear that the biggest expense is salaries, so we can conclude it is a very competitive project since salaries in Barcelona are way lower than those at these leading-research organisations.

Regarding useful life, this project could improve costs since one of the expected results is to be able to use RL for NLP successfully and may reduce the need for data and training time, lowering costs. No big risks can be identi ed for this project, mainly due to its non-pro t nature. The only risk would be a failure of the project, but as we do not expect to obtain bene ts anyway this would only mean a sub-optimal use of resources.

5.2 Environmental Dimension

Regarding the environmental impact of this project, the main resource we will be using is electricity, and in our case we must note that the consumption of the laptop in relation to a GPU clusters in negligible. Training neural networks, is expensive, and even though our training data size is considered small for the eld, it still requires a huge amount of power. Estimating the real emissions of our project is hard although we can try to approximate it. We estimate the nal number of cluster usage hours to be around 300h, so, assuming a carbon footprint of 0.207 kgCO_2/kWh[5] and a power consumption of 0.25kW[6] for the Nvidia GPU:

$$0{:}25kW \quad 300h = 75kWh \quad 75kWh \quad \frac{0{:}207KgCO_2}{1kWh} = 15{:}525KgCO_2$$

To put this consumption into context, it amounts to the same as 75h of Air conditioning A+. Nevertheless, if we take into account that since 2018 the UPC (where our cluster is located) buys 100% renewable energy[7] [8] the carbon

[5] The estimate for the European Environmental Agency for Greenhouse gas emission intensity of electricity generation in Spain in 2019: https://www.eea.europa.eu/data-and-maps/daviz/co2-emission-intensity-6

[6] Sourced from the speci cations of the GPU: https://www.nvidia.com/en-gb/geforce/graphics-cards/rtx-2080-ti/#specs

[7] Link of the press release: https://www.upc.edu/energia2020/ca/noticies/la-upc-contracta-lenergia-electrica-amb-garantia-dorigen-100-renovable

[8] Electricity contracts for both 2020: https://contractaciopublica.gencat.cat/ecofin_pscp/AppJava/awardnotice.pscp?reqCode=viewDcan&idDoc=52978462&lawType=2

footprint is actually close to 0. We assume this has been the real impact of our project, but it must be noted that the calculation of 15.525kg of CO_2 is relevant if we want to generalise this project to an institution that does not use 100% renewable energy.

In order to minimize the carbon footprint of our project (assuming the emissions per kWh are the national average and not the 100% renewable energy of the UPC) the usage of the cluster has to be as minimal as possible. Nevertheless, reusing resources is not possible in our project so the best approach to reducing cluster usage would be to carefully plan the experiments in order to avoid repeating unnecessary executions.

As stated before, this project expectation is to improve state of the art systems so as to require less data and training, and this could reduce environmental impact as well as the possible cost reduction discussed previously.

Finally, since this is a software research project there should be no environmental concerns regarding the useful life or construction and dismantling of the project. The few risks that could increase the carbon footprint would be associated to extending the duration of the project and carrying on more experiments, but this is part of the PPP stage.

5.3 Social Dimension

Undertaking this project in terms of personal growth will provide me the necessary tools to be able to manage a big project. Moreover, it will make me develop work ethic and how it is to work on a big project for a long time. Furthermore, it will introduce me to the world of research and serve as guidance for my future plans.

The accomplishment of the goals of this project could provide a future framework for advances in low-resource Neural Machine Translation that could hugely bene t speakers of minority languages. There is a real need for advances in low-resource languages as it constitutes a step towards equity. Nowadays English works like a *lingua franca* and not having access to English creates a huge disadvantage.

A possible social risk for this project is that our translation systems perpetuate social biases and they get delivered to nal users. This should be a minimal possibility since we will rarely substitute commercial products so as to reach the user, and even if that were the case and our model turned out to be really successful, we assume the normal behaviour would be to then score better on

and 2021: https://contractaciopublica.gencat.cat/ecofin_pscp/AppJava/awardnotice.pscp?reqCode=viewDcan&idDoc=62014818&lawType=3 where it is stated that the electricity utilities will provide 100% renewable energy.

these social biases. Nonetheless, we cannot be sure of that so it constitutes a possible risk.

6 Follow-up

6.1 Tasks completed

From the initial objectives previously described in the initial stage we have completed the following ones:

- Do research in Machine Translation state of the art.
- Do research in Reinforcement Learning techniques.
- Reproduce the state of the art results with the Transformer.
- Compare the proposals with the baseline objectively and rigorously by conducting the necessary experiments and draw conclusions out of the results obtained.

Apart from the objective of documenting all the work, that will be completed eventually, we have completed all the objectives except from one that was modified a bit: *Develop a new system applying RL techniques*. Using RL techniques with the Transformer was not possible for us due to lack of memory of the GPU. Because of that we ended downscaling the architecture to a LSTM Seq-2-Seq model to test the RL techniques. Furthermore, we ended up using the Minimum Risk Training[26] approach instead of the Reinforce one that we previously explained. So in the end, we did not develop a new system and just ended up performing these RL techniques on an extra dataset (FLoRes[9]).

6.2 Deviations from initial planning

In terms of the initial planning phase, we ended up having to delay the project to April due to needing way more extra hours for the "New models development" phase. Up until that task we managed to follow the schedule pretty closely. The code[10] we were supposed to use as starting grounds for our RL algorithms ended up being bugged. We tried contacting the author through Github issues but got no response. We tried fixing the bugs in the code but then we run into another problem: lack of memory in the GPU. We had used all of the 90h anticipated in the risk management section trying to circumvent these bugs and the out of memory of the GPU without success, so we resorted to rescheduling to April. We tried using another library[11], again without success. Finally, as mentioned in the previous subsection, we opted for another library that implemented RL techniques (Minimum Risk Training, effectively a simple

[9] https://github.com/facebookresearch/flores
[10] https://github.com/TianchunH97/fairseq-rl
[11] https://github.com/kokeman/fairseq-RL

variation on REINFORCE, the original approach we were initially going to use as described in section 1.3.1) in a simpler architecture than the Transformer, an alternative already contemplated in the risk management section. To sum up, we ended up using the extra 90h and we rescheduled to April, granting us about 500 extra hours of which we needed about 120h.

6.3 Final cost of the project

There are multiple extra costs that have arisen for the project, mainly due to rescheduling to April. Following the methodology devised in section 4.3:

- Cluster usage deviation: we have required an extra 60h, that amounts to $60h \quad 0{:}42€/h = 25{:}2€$.
- Space: we needed 3 extra months for housing costs (February to April), this amounts to $3\text{months} \quad 420€/\text{month} = 1260€$.
- Laptop amortization deviation: we have required in total 210h extra hours for the laptop, amounting to $210h \quad \frac{1129€}{4\text{ years} \quad 1750h/\text{year}} = 33{:}87€$.
- CPA deviation: of these extra 210h we assume they have been spent as a 65% for the Deep Learning Researcher and 35% for the software developer. Assuming this, we would require $136{:}5h \quad 28{:}79€/h = 3929{:}83€$ extra for the Deep Learning researcher and $73{:}5h \quad 23{:}43€/h = 1722{:}11€$

In total, the project would require an extra of 6971.01€(without taking into account the incidental or the contingency funds). Recall that the total budget for the project was 23627.77€, including 5341.51€ between the incidental and the contingency budgets. We can conclude that the real cost of our project has been 25257.27€ and that the contingency and incidental funds have not been enough and we have needed an extra additional 1629.5€. We have summarised this in table 8.

6.4 Changes in methodology

In terms of tools used in the project, we ended up using an additional library to perform some of our experiments, the Simple-NMT[12] library, that implements some simple algorithms for NMT in PyTorch, including MRT. Furthermore, we also ended up using the FLoReS Nepali-to-English dataset to perform extra experiments apart from the IWSLT14 ones.

6.5 Integration of knowledge

- Evaluation of computational complexity of a problem [Little]: We do not need to understand computational complexity for this project, although it

[12] https://github.com/kh-kim/simple-nmt
[12] https://github.com/facebookresearch/flores

Activity	Cost
Budgeted Personnel Cost by Activity (CPA)	16001.59e
Extra CPA	5651.94e
Real Total CPA	21653.53e
Budgeted Generic Costs (GC)	2284.67e
Extra GC	1319.07e
Real Total GC	3603.74e
Total Extra Costs	6971.01e
Contingency Budget	2742.94e
Incidental Budget	2598.57e
Total Budget	23627.77e
Cost Overrun	1629.50e
Total Real Cost	25257.27e

Table 8: Final total cost of the project.

may be interesting to consider given how computationally complex deep learning is. It is important to try to make linear algorithms as our datasets are huge. On the other hand, it must be noted that analysing the complexity of reinforcement learning algorithms is quite a hard task.

- Knowledge of programming languages [Enough]: As this is a programming project we obviously need to understand and be knowledgeable of programming languages, particularly bash scripting and python programming. It must be noted that by using libraries, we have saved a lot of e ort of programming although at the same time we have required to be knowledgeable in how to use such libraries.

- Capacity to acquire, obtain, formalize and represent knowledge, specially in the case of intelligent systems [Enough]: In NMT it is very important how we represent data, in our speci c case, text. Even though our project does not focus on such encoding, we have to be mindful of it and still use state of the art techniques such as BPE.

- Computational learning [In depth]: The project consists of programming and experimenting with deep learning architectures and paradigms, as such, it is extensively covered. We need understanding of the various deep learning architectures for NMT and also all the Reinforcement Learning theory as well.

- To program taking into account hardware architecture [Enough]: Deep learning needs certain hardware, GPUs, to perform well as it requires a lot of computing power. Precisely, we have encountered problems with excessive GPU memory usage. It is also recommended to adapt your training scripts to better t your GPU and achieve better performance or speeds.

6.6 Identification of applicable laws and regulations

We must respect the licenses of the software we use. In our case: Fairseq uses a MIT License[13], the IWSLT14 dataset is of open use, the FLoRes dataset is under a CC-BY-SA[14] and nally the simple-nmt repository does not specify any LICENSE. In general the software involved in this project is open or with research permissions. Apart from that, legislation does not seem to be applicable in this project.

7 Reinforcement Learning

In this section we will outline the details of the architectures and reinforcement learning algorithms used in the project.

7.1 Architectures

In the rst part of this work we already contextualised the Transformer, however, for the most part of our experiments we ended up using LSTMs with attention to perform the main experiments, as the vanilla Transformer ran out of GPU memory when we tried RL techniques.

As we have mentioned, we ended up using the simple-nmt library for the purpose of implementing the LSTMs as it included a Minimum Risk Training algorithm. As described in its documentation[15], the architecture it implements is a LSTM Sequence-to-Sequence with an Attention Layer as described in the paper by Luong et al.[42]. In this paper they presented a new attention model for NMT at a time where only Bahdanau et al.[18] had successfully applied attention mechanisms to NMT. It must be noted that the simple-nmt library implements a bidirectional LSTM encoder and a LSTM decoder (non-bidirectional). The exact implementation of these LSTMs can be consulted in PyTorch's reference[16].

The aforementioned paper designs two models of attention: *global* and *local*. Common to both of them is the idea that at each time step t of the decoding stage, we take as input the hidden state h_t at the top layer of the stacking LSTM. Our aim is to derive a context vector c_t that captures relevant information in order to predict the current target word y_t. Speci cally, we simply employ a concatenation layer to combine both vectors in order to produce an attentional hidden state $\tilde{h}_t = \tanh(W_c[c_t; h_t])$. We then use this vector to predict the target word as follows $p(y_t|y_{<t}; x) = \text{softmax}(W_s\tilde{h}_t)$. The di erence between these two models of attention is in how this context vector c_t is calculated. We will explain the global attention model since is the one used by the simple-nmt library.

[13] https://github.com/pytorch/fairseq/blob/master/LICENSE
[14] https://github.com/facebookresearch/flores/blob/master/LICENSE
[15] https://github.com/kh-kim/simple-nmt/blob/master/README.md
[16] https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

The idea of the global attention is to consider all the hidden states in order to calculate the context vector, for that purpose we will first derive an alignment vector a_t by comparing the current hidden state h_t with each of source hidden state $\bar{h}_s$. $a_t = \text{align}(h_t; \bar{h}_s) = \text{softmax}(\text{score}(h_t; \bar{h}_s))$.

In order to calculate the score function the paper proposes three alternatives, a dot product, a general one or a concatenation. Simple-nmt uses the general approach: $\text{score}(h_t; \bar{h}_s) = h_t^{\top} W_a \bar{h}_s$.

Given the alignment vector a_t the calculation follows by deriving c_t as the weighted average, according to a_t, of the source hidden states $\bar{h}_s$. To sum up, our calculation path goes as follows $h_t \; ! \; a_t \; ! \; c_t \; ! \; \tilde{h}_t$. We illustrate this model in figure 6. In this proposed model the attentional decisions are made

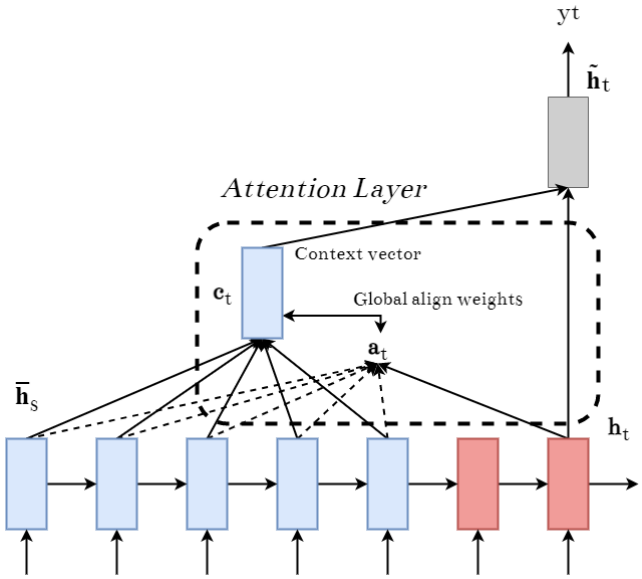

Figure 6: Global attention model: at each time step t, the model derives an alignment vector a_t based on both target, h_t and various sources hidden states, $\bar{h}_s$. A global context vector, c_t is calculated as the weighted average (according to a_t) over all the hidden source states $\bar{h}_s$. Adapted from: [42].

independently which would be suboptimal. For that reason, an *input-feeding* approach is proposed in which we maintain a coverage set by concatenating the attentional vectors $\tilde{h}_t$ with inputs at the next time steps as illustrated in figure 7.

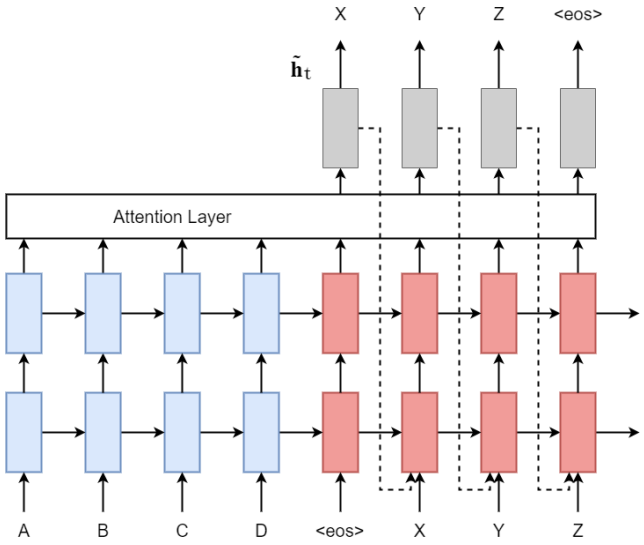

Figure 7: **Input feeding approach**: the attentional vectors $\tilde{h}_t$ are fed as inputs to the next time steps. Adapted from: [42].

7.2 Minimum Risk Training

In Neural Machine Translation there has usually been a mismatch between the loss functions on which we train our models and the actual score (BLEU Score) we expect our models to maximize. In section 1.3.1 we already outlined the idea of applying Reinforcement Learning techniques in order to bridge this gap between loss and real score. The simple-nmt library implements Minimum Risk Training (MRT), that is considered a similar approach to the REINFORCE-based algorithms[25] we explained in that section.

In Minimum Risk Training[26] they de ne a risk function as the expected loss with respect to the posterior distribution:

$$R(\ ) = \sum_{s=1}^{S} E_{y|x^{(s)};\ } [\ (y; y^{(s)})] = \sum_{s=1}^{S} \sum_{y \in Y(x^{(s)})} P(y|x^{(s)};\ )\ (y; y^{(s)}) \quad (1)$$

where $Y(x^{(s)})$ is the set of all possible candidate translations for $x^{(s)}$ and $(y; y^{(s)})$ is a loss function usually a sentence-level evaluation metrics type such as BLEU, NIST, TER or METEOR. In our case, the simple-nmt library implements 4 di erent scoring functions: GLEU score[17][46], and BLEU score

[17] implementation reference of gleu (nltk): https://www.nltk.org/_modules/nltk/translate/gleu_score.html

with three di erent smoothing functions[47]: methods 1, 2 and 4[18]. We have performed our experiments using the default reward: GLEU score.

The implementation of the risk function as in equation 1 faces a major challenge: the search space of $Y(x^{(s)})$ in which we consider all the possible candidate translations is massive and makes calculating the expectations intractable. To alleviate this issue, we only consider a subset of the full set in order to approximate the posterior distribution and we introduce a new training objective:

$$R(\ ) = \sum_{s=1}^{S} \mathrm{E}_{y|x^{(s)};\ ;\ } [\ (y; y^{(s)})] = \sum_{s=1}^{S} \sum_{y \in S(x^{(s)})} Q(y|x^{(s)};\ ;\ )\ \ (y; y^{(s)}) \quad (2)$$

where $S(x^{(s)}) \quad Y(x^{(s)})$ is a sampled subset of the full search space and $Q(y|x^{(s)};\ ;\ )$ is a distribution de ned on such subset as:

$$Q(y|x^{(s)};\ ;\ ) = \frac{P(y|x^{(s)};\ )}{\sum_{y' \in S(x^{(s)})} P(y'|x^{(s)};\ )}$$

note that is a hyper-parameter that controls the sharpness of distribution Q. In the simple-nmt implementation they always assume = 1 thus rendering this parameter useless (as in fact they do not include it).

In practice the simple-nmt library takes random samples based on the multinoulli distribution (multinomial sampling). Unlike the paper in which they used up to 100 samples, the simple-nmt implementation recommends just one sample (similarly to the REINFORCE algorithm), and we have executed our experiments accordingly. The simple-nmt implementation proceeds by taking a sample that we call *actor* and calculating its reward (GLEU score in our case) and storing its $\hat{y}$ (which we will use eventually to calculate logprobs). Then it takes `RL_N_SAMPLES` (a hyper-parameter, in our experiments: 1) samples and averages its rewards into what it calls *baseline* reward. Then, the nal reward is calculated by subtracting this *baseline* from the *actor* reward. Finally, the loss is calculated by multiplying such nal reward by the log probability of the actor sample ($\hat{y}$). Equation 3 re ects the nal loss of our implementation:

$$L(\ ) = \sum_{i=1}^{N} \text{reward}(y_i; \hat{y}_i) \quad \frac{1}{K} \sum_{k=1}^{K} \text{reward}(y_i; \hat{y}_{i;k}) \quad \log P(\hat{y}_i|x_i;\ ) \quad (3)$$

$$\text{where } \hat{y}_i \quad P(y|x_i;\ ):$$

Note how what we called *actor* reward would correspond to the rst reward call and the *baseline* reward would consist of the average on the other reward call.

[18] implementation reference of the di erent BLEU score methods (nltk): `https://www.nltk.org/_modules/nltk/translate/bleu_score.html`

7.3 Complexity

Even though a complete complexity analysis of our models is out of the scope of this project we can make some observations mainly by looking at the complexity analysis from the Transformer's original paper [21] summarized in table 9.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Attention	$O(n^2 \quad d)$	$O(1)$	$O(1)$
Recurrent	$O(n \quad d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \quad n \quad d^2)$	$O(1)$	$O(\log_k(n))$

Table 9: Complexity of various deep learning layers. n is the sequence length, d is the representation dimension and k is the kernel size of convolutions (for CNNs). Source: adapted from [21].

Recall that the Transformer only uses Attention Layers and that our LSTMs models use mainly Recurrent Layers (although they also include an Attention Layer at the top of each stack). Note that the Attention layers outperform the recurrent ones in both sequential operations and maximum path length. The minimum number of sequential operations required indicates that the Transformer can parallelize a higher amount of computations. Additionally, shorter paths mean the model can learn long-range dependencies easier[48].

Moreover, the attention layers will be faster than the recurrent layers whenever the sequence length n is smaller than the representation dimensionality d, which is often the case due to the sentence representations used by most of the state-of-the-art models in machine translation such as BPE.

8 Experimental Framework

In this section we will outline the details of the experiments we have conducted in order to evaluate the aforementioned techniques.

8.1 Initial Considerations

As we have already mentioned it is outside the scope of this project to optimize the hyperparameters of our models, because of that, we will set up the experiments resembling the con gurations of their original papers. In relation to the datasets, we will follow the same approach and use the scripts provided by the datasets authors for data cleaning and splitting. We will compare our models using their own test set.

We will use the default random seed so the experiments will be reproducible.

8.2 Datasets

We will perform our experiments using two di erent corpora: the IWSLT14 DE-EN (German to English) and the FLoReS NE-EN (Nepali to English). Both of these corpora ful ll two features:

- Easily reproducible: it is easy to reproduce the baseline using Fairseq and preprocess and split the data.
- Both of these datasets are considered low-resource, with FLoReS being an extremely low-resource dataset. We want them to be low-resource for two reasons: one would be time of computation and training, as we have limited resources, and the other reason is that it is our goal to apply the RL techniques in low-resource environments to observe if it yields signi cant improvements.

8.2.1 IWSLT14 DE-EN

The IWSLT14 German-English is one of the most used parallel corpus. IWSLT[19] stands for International Workshop on Spoken Language Translation which is a yearly workshop that comprises multiple open tasks in relation to Machine Translation. The IWSLT 14 German-English dataset is composed of translations of TED talks[20] subtitles.

For this dataset, fairseq provides us with a script[21] that splits the data as the original authors of the baseline:

- Train: $\frac{22}{23}$ of the train data provided by IWSLT 14.

[19] https://iwslt.org/
[20] https://www.ted.com/
[21] https://github.com/pytorch/fairseq/blob/master/examples/translation/prepare-iwslt14.sh

- Valid: the rest of the train data provided by IWSLT14 ($\frac{1}{23}$).
- Test: built by combining the dev and test sets of 2010, 2011, and 2012.

In Table 10, we show the exact number of sentences for each of the sets:

Set	# of sentences
Train	160,239
Valid	7,283
Test	6,750

Table 10: Number of sentences for each subset of the IWSLT14 German-English dataset

Even though this task does not have itself that many sentences it is not considered a hard task due to the similarity of German and English (they are both germanic languages).

8.2.2 FLoRes

The Facebook Low Resource (FLoRes)[22] MT Benchmark is a dataset for machine translation between English and 4 low resource languages, Nepali, Sinhala, Khmer and Pashto. It has been sourced from translated sentences from Wikipedia. The authors of the dataset provide the necessary fairseq scripts in order to prepare and preprocess the data as well as to reproduce the baseline. In our experiments we will only use the Nepali-English data. In Table 11 we detail the number of sentences for each subset.

Set	# of sentences
Train	494,878
Valid	2,559
Test	2,835

Table 11: Number of sentences for each subset of the FLoRes Nepali-English dataset

Unlike the aforementioned IWSLT14 and in spite of having more data, this task is really hard as English and Nepali are languages really far apart: they do not even share a common alphabet. Hence the BLEU scores our systems will achieve will be really low.

8.3 Data preprocessing

As we have said, both of this datasets provide preprocessing scripts that tokenize, split into sets and learn BPE from the data. In the case of the IWSLT14

[22] https://github.com/facebookresearch/flores

script[23], the authors use the moses tokenizer[24] and learns a BPE of 10,000 tokens with subword-nmt[25]. On the other hand, in the FLoRes dataset case, as Nepali does not use the latin alphabet, its preprocessing script[26] resorts to using the Indic NLP Library[27] for tokenization, and sentencepiece[28] for learning a BPE of 5,000 tokens.

8.4 Baselines

We planned for the vanilla Transformer to be our baseline, however, due to not being able to perform the RL experiments we have to include a new baseline, an LSTM with Attention. Nonetheless, we will experiment with the vanilla Transformer as well and include those results.

For the German-English Transformer case, we use the architecture [21] described by the original paper that it is implemented in Fairseq with the hyperparameters proposed by the Fairseq authors[29]. Particularly, the architecture[30] is composed of 6 layers for both the encoder and decoder, 4 attention heads, 512 embedding size and 1024 size for the feedforward, they also use a dropout of 0.3, label smoothing of 0.1, 4096 maximum tokens per batch and a starting learning rate of 5e-4.

In the case of the FLoRes Transformer we use the hyperparameters proposed in their repository[31], that is, 5 layers for both the encoder and decoder, 2 attention heads for both of the encoder and decoder, 512 embedding size, 2048 feed-forward size, 0.4 of dropout, 0.2 of label smoothing, a maximum of 4000 tokens per batch, and a starting learning rate of 1e-3.

The LSTM baselines we have used may not be that accurate as the Transformers' ones because we do not have a clear reference unlike with the Transformer where we had a fairseq recommendation. Furthermore, for these baselines we used Fairseq only for preprocessing and scoring, and we have used the simple-nmt library for training, at least for the German-English case we can try to resemble the hyperparameters described in the architecture paper [42], where they also experiment with a German-English dataset, although their dataset is much larger and they mainly focus on the English to German translation (the inverse of our task). We also tried a model similar to the parameters the author

[23] https://github.com/pytorch/fairseq/blob/master/examples/translation/prepare-iwslt14.sh

[24] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/tokenizer/tokenizer.perl

[25] https://github.com/rsennrich/subword-nmt

[26] https://github.com/facebookresearch/flores/blob/master/prepare-neen.sh

[27] https://github.com/anoopkunchukuttan/indic_nlp_library

[28] https://github.com/google/sentencepiece

[29] https://github.com/pytorch/fairseq/tree/master/examples/translation

[30] https://github.com/pytorch/fairseq/blob/ee0d5a0f65a25e5f5372776402aac5cb9c4adbf1/fairseq/models/transformer.py#L1132

[31] https://github.com/facebookresearch/flores#train-a-baseline-transformer-model

of simple-nmt uses for their experiments with the Korean-English language pair.

For both the German-English and the FLoReS pair we used 4 layers, a batch size of 128, max length of 100, dropout of 0.2, word vector size of 512, and a hidden layer size of 768, a starting learning rate of 1e-3, and a max gradient norm of 1e8.

In table 12 we outline a summary of the hyperparameters used.

	Seq2Seq		Transformer	
Parameter	IWSLT14	FLoRes	IWSLT14	FLoRes
Layers	4	4	6	5
Attention heads	-	-	4	2
Embedding/wordvec size	512	512	512	512
Ffn size/hidden size	768	768	1024	2048
Dropout	0.2	0.2	0.3	0.4
Starting learning rate	1e-3	1e-3	5e-4	1e-3
Batch size	128	128	-	-
Max tokens	-	-	4096	4000
Clip norm/max gradient norm	1e8	1e8	0	0

Table 12: Relevant hyperparameters for reproducing the baselines.

8.5 Other details

As we have already mentioned our models have been built using Fairseq and simple-nmt. Particularly, Fairseq is the framework used by the Machine Translation group, in which this work has been developed. Fairseq was also chosen because it is backed by Facebook AI and frequently updated, including state-of-the-art machine translation components. Nevertheless, it does not implement any sort of reinforcement learning (even though it does stem from MIXER[32] that implements a reinforce-like algorithm but is now archived and not updated frequently), and there are not many forks that do so. Our first approach was to use those repositories in order to be able to keep all the models within the fairseq framework but this was not possible in the end. Because of that we resorted to using the simple-nmt library too. It is worth mentioning that both this libraries are based on PyTorch so even though they are not directly compatible, they are not hard to combine.

For an implementation reference and scripts to reproduce the experiments refer to appendix A. The source code will be made available in Github[33].

One part of the project that is not reflected in the results but ended up consuming a lot of time was the debugging of the main RL repository[34] we were

[32] https://github.com/facebookarchive/MIXER
[33] https://github.com/jmuntaner/tfg
[34] https://github.com/TianchunH97/fairseq-rl

going to use. The code provided was missing key modules that we added and then it contained deprecated parts of code that we tried to update. We did not succeed in fixing the code as we would end up encountering the out of memory bug. In fairseq the loss function is managed by the criterion class[35], where we can alter and obtain information from the forward pass in order to calculate our loss function.

8.6 Proposed experiments

The idea is to experiment with the following combinations:

- Datasets: IWSLT and FLoRes
- Architectures: Transformer and bi-LSTM
- Loss functions: MLE and MRT

If we had extra time we could also include hyperparameters to the experiments, such as embedding sizes or learning rates, nevertheless it is not the main goal of this project.

[35] https://fairseq.readthedocs.io/en/latest/criterions.html

9 Results

In this section we are gonna present all of the results obtained in our experiments.

9.1 IWSLT14 DE-EN

We started reproducing the transformer baseline for the IWSLT14 DE-EN dataset, we can observe the results obtained in table 13.

Set	Loss
Training	3.166
Validation	3.844

Set	BLEU
Test	35.07

Table 13: Results of the transformer architecture in IWSLT14 German-English.

It trained for 46 epochs reaching the lowest validation loss at the 40th epoch and remained stable for the remaining epochs, this suggests that the training converged. In gure 8 the evolution of the train loss and validation loss are depicted. The best validation perplexity was 4.92 on the 40th epoch as well.

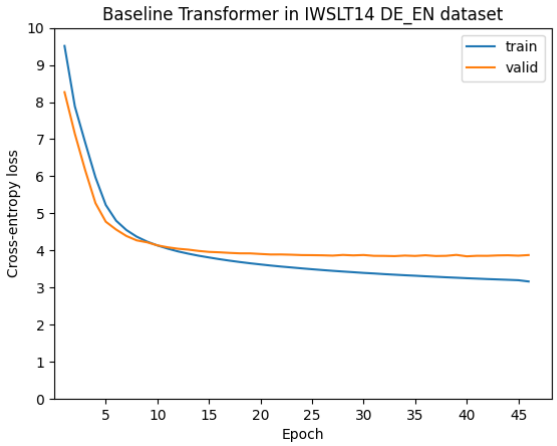

Figure 8: Evolution of validation and train losses during training of the IWSLT14 transformer baseline. It can be observed how when the training stopped, we had already achieved the best validation loss, hence the remaining improvement in train loss was due to over tting.

9.1.1 IWSLT14 with Bi-LSTM

We performed di erent experiments with the LSTMs architecture. Firstly we tried replicating a baseline of the simple architecture with MLE loss. We ex-

perimented with two di erent set of parameters, the best result obtained used the parameters described in section 8.4. The results of the rst model are summarised in table 14.

Set	Loss
Training	0.821
Validation	1.478

Set	BLEU
Test	29.96

Table 14: Results of the bi-LSTM architecture in IWSLT14 German-English using MLE.

In the simple-nmt library we indicate the number of training epochs, in our case, 30 epochs, and as one can tell by the evolution of the loss function represented in gure 9, the validation loss reached its best value at the 13th epoch and stayed mostly at (even a little worse due to over tting to the training data) for the remaining epochs indicating convergence.

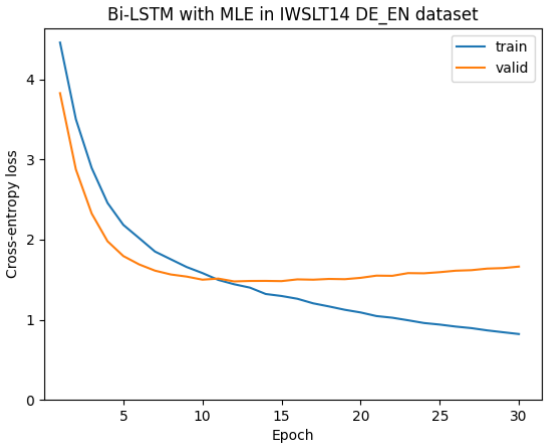

Figure 9: Evolution of validation and train losses during training of the IWSLT14 bi-LSTM baseline. It can be observed how when the training stopped, we had already achieved the best validation loss, hence the remaining improvement in train loss was due to over tting which actually caused a worse validation loss.

As we have said we also experimented with di erent parameters but got a BLEU score a bit lower (29.93). We changed the hidden size to 1024, the max_length to 50, the max_grad_norm to 5 and lr_start_decay to 12. As we can tell by the loss graph in gure 10, this training was much more spiky than with the other parameters.

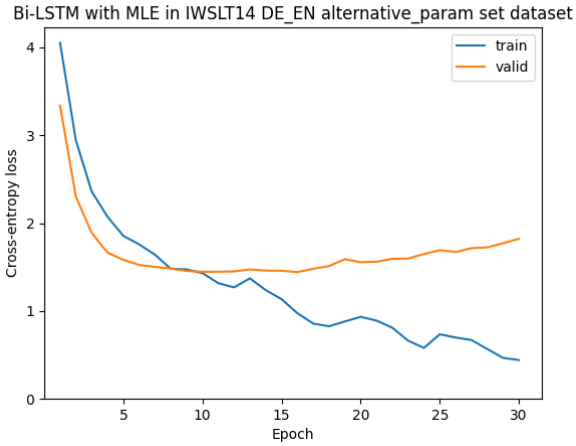

Figure 10: Evolution of validation and train losses during training of the IWSLT14 bi-LSTM baseline (using MLE loss). it is noticeable how this graph is much less smoother than the loss graph produced by the other parameters in gure 9

.

9.1.2 Reinforcement Learning

Next we trained the same architecture using the Minimum Risk Training loss[36]. We used the model with the best validation loss from the mle training and continued training for 10 epochs using the RL criterion. We obtained a signi cant improvement as the new model achieved an improvement of nearly a BLEU point. Results are detailed on table 15.

Set	BLEU Loss
Training	39.83
Validation	45.44

Set	BLEU
Test	30.79

Table 15: Results of the bi-LSTM architecture in IWSLT14 German-English dataset after training with MRT.

We can observe the evolution of the loss (in this case it is BLEU Score) as the model trained in gure 11.

Looking at the loss graph it was not clear if the training had converged or

[36] It must be noted that for all the RL experiments we used the parameters provided on the simple-nmt example, that is `1 iteration_per_update` and a `max_grad_norm` of 5.

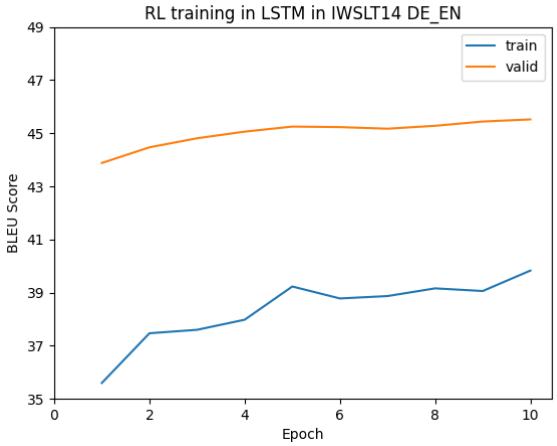

Figure 11: Evolution of the MRT BLEU Score as it trains on the IWSLT14 DE-EN data for 10 epochs.

found a local minimum, for that reason we repeated the experiment but this time we left it training for 40 epochs, obtaining the following results as exposed in table 16.

Set	BLEU Loss
Training	42.94
Validation	45.97

Set	BLEU
Test	31.2

Table 16: Results of the bi-LSTM architecture in IWSLT14 German-English dataset after training with MRT for 40 epochs.

We can observe that we improved around 0.4 BLEU points by letting it trained longer. Is palpable mainly in the increase of the BLEU losses (more is better here). We can see in gure 12 how the training was a bit irregular but reached its best validation BLEU score at the 31th epoch.

9.1.3 Summary of IWSLT14

Finally, in table 17 we can see a comparison between all the models mentioned. We can observe that even though the MRT training yields a signi cant improvement (1.24 BLEU points), we are still far from reaching the score obtained by the Transformer.

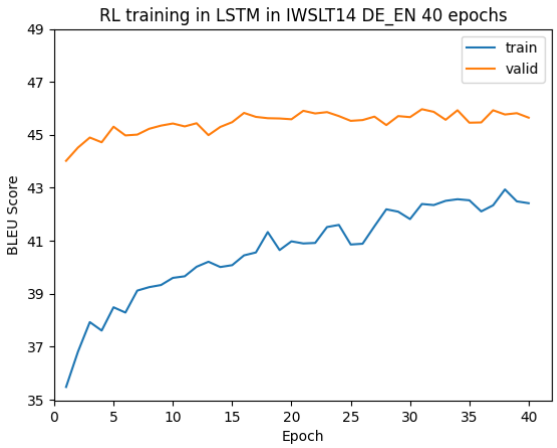

Figure 12: Evolution of the MRT BLEU Score as it trains on the IWSLT14 DE-EN data for 40 epochs.

Model	BLEU Score
Vanilla Transformer MLE	35.09
Bi-LSTM MLE + MRT Training	31.20
Bi-LSTM MLE	29.96
Bi-LSTM MLE alt. param.	29.93

Table 17: Summary of all the results obtained in the IWSLT DE-EN dataset.

9.2 FLoRes NE-EN

Next we performed the same series of the experiments but this time using the FLoRes dataset with the goal of seeing if this improvement translated to harder tasks. As before, we first trained a model using the Transformer and MLE loss. We can see these results in table 18 where we can already appreciate that this task is much harder since we only get 3.19 BLEU score, much lower than what we obtained with the previous dataset.

Set	Loss
Training	3.947
Validation	8.3

Set	BLEU
Test	3.19

Table 18: Results of the transformer architecture in the Nepali-English dataset.

The model trained for a 100 epochs reaching the lowest validation loss of 8.3 at the 84th epoch although it was already stabilised around 8.3-8.4 since epoch 25th, so we can assume the remaining epochs did not really achieve much. We can see this depicted in figure 13 where both loss curves are almost flat by the

25th epoch. The lowest validation perplexity was pretty high at 85.28 at the 54th epoch but again since around epoch 25 it oscillated around 85 and 95.

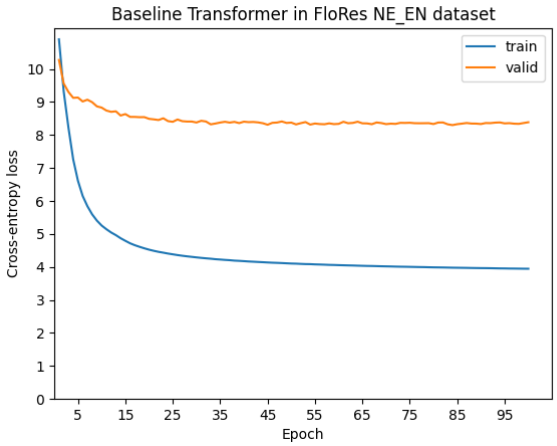

Figure 13: Evolution of validation and train losses during training of the transformer with the FLoRes data. The atness that starts by epoch 25 tells us the model converged approximately there.

9.2.1 FLoRes with Bi-LSTM

In the same way as with the IWSLT14 we tested the Bi-LSTM architecture on the FLoRes NE-EN dataset. First of all we trained a baseline for 30 epochs with the parameters described in section 8.4. We expose the results in table 19.

Set	Loss
Training	0.298
Validation	4.69

Set	BLEU
Test	0.73

Table 19: Results of the bi-LSTM architecture in the Nepali-English dataset using MLE loss.

We depict the evolution of the loss in gure 14. The best validation loss was achieved at the second epoch and then got worse increasingly, as we can see in the graph. This indicates us that the architecture was not really able to learn much and started over tting to the training data right away.

For that reason, we tried training with a di erent set of hyperparameters, similar to the ones we tried on with the DE-EN dataset. We changed the `hidden size` to `1024`, `dropout` to `0.4`, `max_grad_norm` to `5` and `lr_decay_start` to `12`.

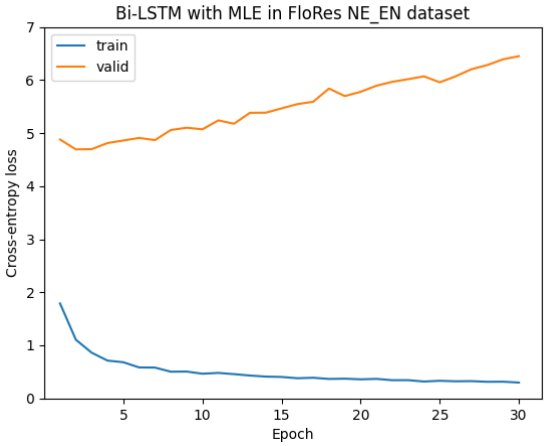

Figure 14: Evolution of validation and train losses during training of the bi-LSTM using the FLoRes NE-EN data. The constant increase in validation loss and decrease in training loss indicates over tting.

With this parameters we managed to achieved "signi cant" training until the 5th epoch. We expose the results this model yielded in table 20 and depict the loss evolution graph in gure 15.

Set	Loss
Training	0.261
Validation	4.527

Set	BLEU
Test	1.74

Table 20: Results of the bi-LSTM architecture in the Nepali-English dataset using MLE loss and alternative parameters.

Surprisingly, this model scored approximately a whole point more than the other version.

9.2.2 Reinforcement Learning

Even though the rst bi-LSTM model we trained was not really good, as it just improved for an epoch before over tting, we thought it would be interesting to see how MRT training would do when the base model is sub-optimal. We trained a MRT[37] taking that second best epoch as the base model for 40 epochs. We saw some improvement as it managed to reach 1.10 BLEU Score, however it is still lower than the other baseline. We expose the results of this model in

[37] It must be noted that for all the RL experiments we used the parameters provided on the simple-nmt example, that is `1 iteration_per_update` and a `max_grad_norm` of 5.

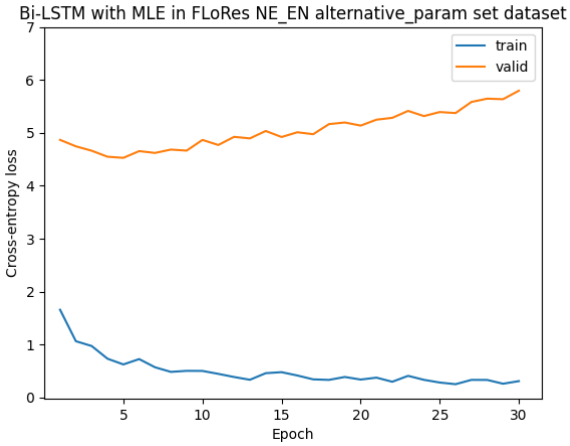

Figure 15: Evolution of validation and train losses during training of the bi-LSTM using the FLoRes NE-EN data. This model, unlike the previous one, managed to improve validation loss until epoch 5th where it started to over t again.

table 21 and the evolution of the loss function in gure 16.

Set	Training BLEU Score
Training	60.92
Validation	6.53

Set	BLEU
Test	1.10

Table 21: Results of the bi-LSTM architecture in the Nepali-English dataset using MRT starting from the worse model.

The other experiment we performed with MRT[38] with this dataset is similar to the previous one with the di erence of using a better base model for the MRT training. We expose the results on table 22 and the BLEU Score evolution on gure 17.

Set	Training BLEU Score
Training	69.55
Validation	8.16

Set	BLEU
Test	1.98

Table 22: Results of the bi-LSTM architecture in the Nepali-English dataset using MRT starting from the best baseline.

[38] It must be noted that for all the RL experiments we used the parameters provided on the simple-nmt example, that is `1 iteration_per_update`

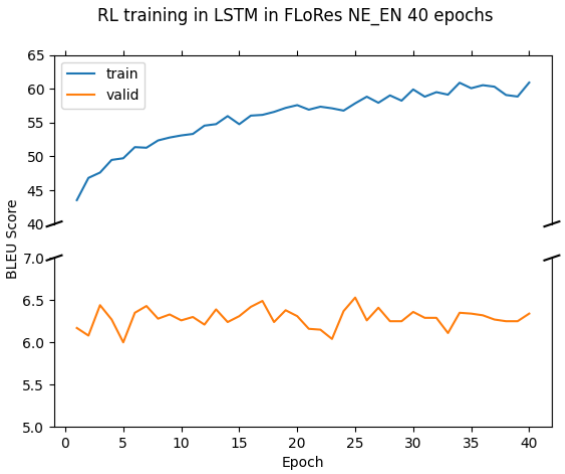

Figure 16: Evolution of validation and train BLEU Score during training of the bi-LSTM using the FLoRes NE-EN data. We can appreciate how the validation BLEU Score follows an irregular pattern and does not clearly converge while the train BLEU score keeps growing (which indicates over tting to the train data). It is also noticeable the big gap in scores between the train and validation sets, this is probably due to how the sets are formed: the train set is mostly composed of single words or really short sentences while the validation set is entirely made up of sentences, obtaining high BLEU Scores for single word translations is usually much easier.

9.2.3 Summary of FLoRes

Finally, in table 23 we can see a comparison between all the models mentioned.

Model	BLEU Score
Vanilla Transformer MLE	3.19
Bi-LSTM MLE alt. param. + MRT Training	1.98
Bi-LSTM MLE alt. param.	1.74
Bi-LSTM MLE + MRT Training	1.10
Bi-LSTM MLE	0.73

Table 23: Summary of all the results obtained in the FLoRes NE-EN dataset.

9.3 Transformer and Reinforcement learning

As we have mentioned already, the initial goal of this project was to perform these experiments using the Transformer architecture, however we encountered a CUDA OOM problem that prevented us from doing so. The Transformer model, although it performs better than the LSTMs ones, consumes much more

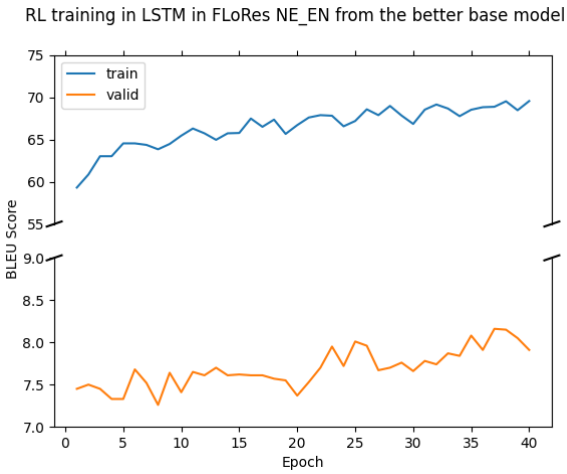

Figure 17: Evolution of validation and train BLEU Score during training of the bi-LSTM using the FLoRes NE-EN data using MRT taking the better base model.

memory per batch. We tried running it with both the libraries we used, and we had to get the batch size as low as 8 for the simple-nmt to not get an OOM error. For the case of the Fairseq library, even getting as low as 200 max_tokens we eventually ran out of memory and the model stopped training quickly. Getting lower did not allow us to include full sentences. We assume that with a GPU with more memory we could perform these experiments correctly.

10 Conclusions

General discussion In this project we began from a few works recent works that indicated that Reinforcement Learning could be useful to improve NMT systems by bridging this dissonance between the two metrics, the training loss, and the BLEU Score, and how we want the best BLEU Score, not necessarily the best loss. Reinforcement Learning can be really costly, and the few previous works on it have been using large datasets, so we thought it would be interesting to experiment in low-resource environments.

The initial goal of the project was to explore the di erent possibilities of Reinforcement Learning for NMT and try to apply it to the state-of-the-art architecture, the Transformer, and maybe try to develop some improvements upon it. However, we ended up facing a lack of GPU memory as the model was too demanding. This particular matter also costed the project a lot of unplanned time to try to x the issue, however it did not amount to any result.

In the face of this problem, we shifted the approach of this work towards a more simplistic one, and perform a di erent set of experiments and try to gain insight of how RL fares up with recurrent models. In order to do that, we resorted to the simple-nmt library because of its simplicity and its already implemented MRT training, that even though it is not exactly the same as the Reinforce algorithm we planned on using, it is considered to be an slight variation.

We found out that Reinforcement Learning yields signi cant improvements over the baseline model, in the IWSLT14 Dataset we obtained an improvement of 1.24 BLEU points, that should not be possible due to "luck", as we did not observe a high variability between models (furthermore, there was only a 0.16 di erence between di erent set of parameters). It supposes an increase of about a 4% of the BLEU score respective to the training without RL.

We then replicated the same experiments with the harder task FLoRes NE-EN to see if it also yielded a similar improvement. In this case the absolute improvement was smaller (around 0.24 BLEU points), but it amounted to a 13.8% improvement. We also experimented with another parameters that yielded a lower result, this time obtaining an improvement of 0.37 BLEU points, an improvement of around 50% in the BLEU Score. This last particular result is in line with [25] where they outline that Reinforcement Learning actually works as a sort of ne-tuning.

We consider this project was really ambitious in their initial goals, as we tried to improve a state-of-the-art result in a eld that is quite "new". As far as we know there were not many works applying Reinforcement to NMT e ectively when we started the project[33]. In fact, amongst the few papers related to this topic some like [25] state that reinforcement learning does not actually produce

meaningful results as it is now. Reinforcement Learning for NMT is a whole research line and it is a fairly "new" one which makes improvements really hard. In the end, and as we have already mentioned, we could not fulfill our ambitious goals and had to set up for a simpler approach.

To sum up, the main contributions of this work are: training the Transformer with a RL technique unsuccessfully, which indicated that such model requires high amount of GPU memory when applying RL training; training (this time successfully) a bi-LSTM network with a simple variation of the same RL technique we tried on the Transformer; experimenting with extremely low-resource datasets (IWSLT14 DE-EN, FLoRes NE-EN) and ascertaining that the RL technique still yields a significant improvement over the baseline (of not using RL training) in these low-resource environments.

Personal conclusions In relation to personal matters, I have learned various skills in the course of this project. I have learned about managing a big project, particularly a research one. Furthermore, I have learned how to tackle potential obstacles in your planning and look for alternatives. To be fair this project faced a lot of adversity as we had to drop the experiments with the RL Transformer because it was not leading anywhere. Technically, I have appreciated how it is fairly easy to use deep learning for well-studied tasks as libraries such as PyTorch or FairSeq make it straight-forward. However, if one wants to change something about such libraries or test a new system, it can get really difficult to do so, specially for research matters, as sometimes it is hard to discern between an error or a bad result. Additionally, this project has given me insight into the research world and how it is to be part of a research group, attending the group seminars provided me with a lot of knowledge of machine translation in general.

Future work As we have already said there are few works that explore reinforcement learning in NMT. Some options that we could try are:

- Gated Transformer-XL [44]: In this paper, they develop a new architecture in order to adapt the Transformer to Reinforcement Learning setups. Maybe we could adapt this architecture to the NMT case and test different RL methods.
- Exploring Supervised and Unsupervised rewards [43]: in this paper they adapt the Soft-Actor Critic framework to the NMT and successfully explore unsupervised rewards for NMT. Using their rewards functions would be an interesting step.
- Using monolingual data: as proposed in [34], we could use monolingual data to boost the performance of our models. Even a more complex approach would be to train generative networks to create such monolingual data for us.

- Adapting diversity-based methods [45] to the NMT context, in this work they propose a method for reinforcement learning without supervision. In a number of experiments their method manages to solve the task without actually learning a reward. However we assume this option to be really hard.

In relation to experimentation, we suggest:

- Experimenting with a REINFORCE loss function and compare the results to the ones we have obtained. This way, we would be able to confirm by ourselves that our MRT-based models will not differ much from REINFORCE-based models.
- Trying different configurations of hyperparameters in order to optimise them. This is the most straight forward improvement we could do, trying to search for the best hyperparameters and trying new configurations of them.
- Evaluating our models in out of domain data: both of our datasets are a bit specific (IWSLT is based of TED Talks, and the FLoRes dataset is composed of Wikipedia translations. It would be interesting to see how our models would fare up to our of domain data.
- Testing news datasets, for example, the FLoRes dataset includes 3 more languages that we could test on. Furthermore, we have only experimented with English as the target language. We could include experiments with English as the source language.
- Performing an extensive study on complexity, memory usage and performance of the models using profiling tools. This way, we could further confirm the high cost of using RL techniques with the Transformer architecture.

References

[1] Machine translation upc research group webpage. mt.cs.upc.edu.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[5] Kyunghyun Cho. Introduction to neural machine translation with gpus (parts 1, 2 and 3). https://developer.nvidia.com/blog/introduction-neural-machine-translation-with-gpus/.

[6] Rohit Mundra, Francois Chaubard, Michael Fang, Guillaume Genthial, Richard Socher, Lucas Liu, Barak Oshri, Kushal Ranjan, Milad Mohammadi, Lisa Wang, Amita Kamath, Amani Peddada, Qiaojing Yan, Emma Peng, and Ajay Sohmshetty. Cs224n: Natural language processing with deep learning lecture notes parts i-vi. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/.

[7] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263{311, 1993.

[8] Antonio Valerio Miceli Barone, Barry Haddow, Ulrich Germann, and Rico Sennrich. Regularization techniques for ne-tuning in neural machine translation, 2017.

[9] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks, 2019.

[10] Yann Lecun. A theoretical framework for back-propagation. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, pages 21{28. Morgan Kaufmann, 1988.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[12] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[13] Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735{1780, 1997.

[14] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks, 2014.

[15] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the di culty of training recurrent neural networks, 2013.

[16] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[17] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[18] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[19] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311{318, USA, 2002. Association for Computational Linguistics.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Je rey Dean. E cient estimation of word representations in vector space, 2013.

[21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[22] Vinod Nair and Geo rey E. Hinton. Recti ed linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807{814, Madison, WI, USA, 2010. Omnipress.

[23] Jimmy Lei Ba, Jamie Ryan Kiros, and Geo rey E. Hinton. Layer normalization, 2016.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[25] Leshem Choshen, Lior Fox, Zohar Aizenbud, and Omri Abend. On the weaknesses of reinforcement learning for neural machine translation, 2019.

[26] Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum risk training for neural machine translation, 2016.

[27] Samidh Chatterjee and Nicola Cancedda. Minimum error rate training by sampling the translation lattice. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 606{615, Cambridge, MA, October 2010. Association for Computational Linguistics.

[28] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 278{287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[29] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction, 2017.

[30] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks, 2016.

[31] Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *Proceedings of the Seventeenth Conference on Uncertainty in Arti cial Intelligence*, UAI'01, page 538{545, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[32] Tianchun Huang. A study of improving bleu using rl loss in nmt. `https://github.com/TianchunH97/fairseq-rl/blob/master/A_Study_of_RL_loss.pdf`.

[33] Julia Kreutzer. Rl in nmt: the good, the bad and the ugly. `http://www.cl.uni-heidelberg.de/statnlpgroup/blog/rl4nmt/`, 2018.

[34] Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. A study of reinforcement learning for neural machine translation. *CoRR*, abs/1808.08866, 2018.

[35] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[36] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229{256, May 1992.

[37] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Kuttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Ga ney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft ii: A new challenge for reinforcement learning, 2017.

[38] Richard Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst*, 12, 02 2000.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024{8035. Curran Associates, Inc., 2019.

[40] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[41] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, and M. Federico. Report on the 11 th iwslt evaluation campaign , iwslt 2014. 2015.

[42] Thang Luong, Hieu Pham, and Christopher D. Manning. E ective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412{1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[43] Julia Ive, Zixu Wang, Marina Fomicheva, and Lucia Specia. Exploring supervised and unsupervised rewards in machine translation, 2021.

[44] Emilio Parisotto, H. Francis Song, Jack W. Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant M. Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, Matthew M. Botvinick, Nicolas Heess, and Raia Hadsell. Stabilizing transformers for reinforcement learning, 2019.

[45] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function, 2018.

[46] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Je Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cli Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macdu Hughes, and Je rey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

[47] Boxing Chen and Colin Cherry. A systematic comparison of smoothing techniques for sentence-level BLEU. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 362{367, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.

[48] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber. Gradient ow in recurrent nets: the di culty of learning long-term dependencies. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press, 2001.

A Implementation reference

In this appendix we provide an overview of our scripts and some examples.

A.1 Overview

We recommend reading the Fairseq documentation[39] as well as taking a look at both Fairseq[40] and simple-nmt[41] repositories in order to have a better understanding of our code. It must be noted that our scripts are intended for running in a GPU Cluster with Slurm.

The relevant files in our code are:

- `./fairseq/fairseq/criterions/v2.py`: file that implements the REINFORCE algorithm, it is derived from Tianchun's github[42] with a few modifications.
- `./fairseq/examples/translation/prepare-iwslt14.sh`: script that prepares the IWSLT14 DE-EN data (download, tokenization, BPE learning, set splitting).
- `./fairseq/flores/download-data.sh`: script that downloads the data from the FLoRes dataset.
- `./fairseq/flores/prepare-neen.sh`: script that prepares the FLoRes NE-EN data (tokenization, BPE learning, set splitting).
- `./fairseq/train_baseline_smoothed.sh`: script for training a baseline using the Transformer with label-smoothing with the IWSLT14 DE-EN dataset.
- `./fairseq/generate_baseline_smoothed.sh`: script that performs inference and BLEU scores the model obtained by the previous script.
- `./fairseq/train_baseline_neen.sh`: script for training a baseline using the Transformer with label-smoothing with the FLoRes dataset.
- `./fairseq/generate_neen.sh`: script that performs inference and BLEU scores the model obtained by the previous script.
- `./experiments/utils_preprocessing/nlp_preprocessing/detokenizer.py`: script that detokenizes data, useful for when we score the FLoRes dataset.
- `./experiments/plots`: scripts for parsing training log files of simple-nmt, and generating graphs of the loss function. Adapted from this file[43].

[39] https://fairseq.readthedocs.io/en/latest/
[40] https://github.com/pytorch/fairseq
[41] https://github.com/kh-kim/simple-nmt
[42] https://github.com/TianchunH97/fairseq-rl
[43] https://github.com/jordiae/fairseq-factored/blob/master/plot/plots.py

- `./fairseq/plots`: scripts for parsing training log files of fairseq, and generating graphs of the loss function. Adapted from this file[44].
- `./experiments/simplenmt/simple-nmt/evaluate_generic.sh`: generic script that performs inference on IWSLT14 models.
- `./experiments/simplenmt/simple-nmt/evaluate_generic_flores.sh`: generic script that performs inference on FLoRes models.
- `./experiments/simplenmt/simple-nmt/score_generic.sh`: script that BLEU scores the translations obtained by the `evaluate_generic.sh` script, that is, from the IWSLT14 dataset.
- `./experiments/simplenmt/simple-nmt/score_generic_flores.sh`: script that BLEU scores the translations obtained by the `evaluate_generic_flores.sh` script, that is, from the FLoRes dataset.
- IWSLT14 training scripts, located at `./experiments/simplenmt/simple-nmt/`:
  - `deen_train_mle.sh`: trains a model with Bi-LSTM using MLE loss.
  - `deen_train_mle_alt.sh`: trains a model with Bi-LSTM using MLE loss and an alternative set of parameters.
  - `deen_train_mrt.sh`: trains a model with Bi-LSTM using MRT loss.
- FLoRes training scripts, located at `./experiments/simplenmt/simple-nmt/`:
  - `neen_train_mle.sh`: trains a model with Bi-LSTM using MLE loss.
  - `neen_train_mle_alt.sh`: trains a model with Bi-LSTM using MLE loss and an alternative set of parameters.
  - `neen_train_mrt.sh`: trains a model with Bi-LSTM using MRT loss.
  - `neen_train_mrt_alt.sh`: trains a model with Bi-LSTM using MRT loss starting from the alternative MLE model.

A.2 Usage

First of all we need to fulfill the requirements to be able to execute the code, (see `requirements.txt`. Then we need to download and prepare the data. For the IWSLT14 case we proceed by:

```
# Download and prepare the data
cd fairseq/examples/translation/
bash prepare-iwslt14.sh
cd ../..

# Preprocess/binarize the data
TEXT=examples/translation/iwslt14.tokenized.de-en
fairseq-preprocess --source-lang de --target-lang en \
```

[44] https://github.com/jordiae/fairseq-factored/blob/master/plot/plots.py

```
    --trainpref $TEXT/train --validpref $TEXT/valid --testpref
    $TEXT/test \
    --destdir data-bin/iwslt14.tokenized.de-en \
    --workers 20
```

Listing 1: IWSLT14 data preparation

For the FLoRes data we execute (from within the ores folder in fairseq (`./fairseq/flores`):

```
bash download-data.sh
bash prepare-neen.sh
```

Listing 2: FLoRes data preparation

A.2.1 Training a Transformer model

In order to training a Transformer model we proceed by executing the following

```
sbatch train_baseline_smoothed.sh
```

We have to take into account that as we are executing this models in a slurm cluster, the output will appear in the le we set as output in the line `#SBATCH --output=`. We also can choose the location of the checkpoints of the training by changing the `CP_DIR` variable. Finally, it is also important to set the Fairseq directory correctly into the variable `FAIRSEQ_DIR`. Once we have nished training, we can perform inference and scoring of our best model (located in the checkpoints folder we speci ed) by executing(again, we must be mindful of the checkpoints, fairseq and output directories and les:

```
sbatch generate_baseline_smoothed.sh
```

By now, we can check the output le and see what BLEU score our model has achieved.

A.2.2 Training a simple-nmt model

The pipeline for training the models here is `train -> evaluate -> score`. We will show how to train an MLE model and train MRT upon it. In order to execute the train scripts properly, as before one has to assure to input the correct directories for the data, the model destination folder and the output log le. For example, lets train a FLoRes model:

```
sbatch neen_train_mle_alt.sh
```

Now, looking at the output log le we can select the best model (in our case the one with the lowest validation loss) and perform inference. We can use the `evaluate_generic_flores.sh` script for that purpose, we only need to specify the model path and the output le for the translated sentences.

```
sbatch evaluate_generic_flores.sh
```

By now, we already have our hyps, we just need to detokenize the sentences and compare them to the references in order to get a BLEU Score. We use the `score_generic_flores.sh` script for that purpose. We have to modify the

HYPS_DIR variable so that it points to the translated sentences file we obtained from the evaluate script. Note that we have to adequately indicate the Fairseq and preprocessing directories as well as making sure our environments fulfill all requirements[45].

```
sbatch score_generic_flores.sh
```

At this point, we could check the obtained BLEU Score in the output log file. Now if we wanted to train a MRT model upon our trained MLE model, we simply would have to follow again with the `train -> evaluate -> score` pipeline, although this time we should use the mrt version of the train script (neen_train_mrt_alt.sh).

[45] The script uses two different conda environments because in our settings the simple-nmt library and the fairseq library were in different folders under different conda environments.