Findings of the First Shared Task on Lifelong Learning Machine Translation

Loïc Barrault
University of Sheffield
l.barrault@sheffield.ac.uk

Magdalena Biesialska & Marta R. Costa-jussà
Universitat Politècnica de Catalunya
first.last@upc.edu

Fethi Bougares
LIUM
fethi.bougares@univ-lemans.fr

Olivier Galibert
LNE
Olivier.Galibert@lne.fr

Abstract

A lifelong learning system can adapt to new data without forgetting previously acquired knowledge. In this paper, we introduce the first benchmark for lifelong learning machine translation. For this purpose, we provide training, lifelong and test data sets for two language pairs: English-German and English-French. Additionally, we report the results of our baseline systems, which we make available to the public. The goal of this shared task is to encourage research on the emerging topic of lifelong learning machine translation.

1 Introduction

Lifelong learning can be defined as the ability to continually acquire new and retain previous knowledge. This ability characterizes humankind, but it is also reflected in several artificial intelligence systems (Parisi et al., 2019; Biesialska et al., 2020). There are many challenges that have to be solved in order to achieve this goal of continual adaptation, among which catastrophic forgetting (French, 1999) seems to be the most relevant.

Lifelong learning is very useful in the area of machine translation (MT), as it allows MT systems to adapt to new vocabularies and topics, and produce accurate translations across time. Currently, there are no previous works that systematically try to solve the problem. This may be due to the lack of a benchmark to address the challenge (Biesialska et al., 2020).

In this context, the main goal of the shared task on lifelong learning for MT is to develop systems that can self-adapt relying solely on domain expert data and are then freed from the necessity of machine learning expertise. What is more, this shared task also allows to investigate several MT research directions, such as: the continuous training/adaptation techniques; the preparation of additional publicly available corpora and evaluation sets; the active learning methods via a controlled simulated environment; the unsupervised adaptation of MT systems; the document-level approaches and the development and evaluation of MT systems across time.

2 Related work and tasks

As mentioned in the introduction, there are not really any works in MT properly evaluating lifelong learning systems. However, there is a long history of studies in related tasks that are useful for addressing the lifelong learning objective.

Domain adaptation is based on the premise that the system can adapt to a target domain known in advance. This has been widely studied earlier for statistical MT e.g. (Koehn and Schroeder, 2007) and, more recently, for neural MT e.g. (Luong and Manning, 2015)).

Instance-based adaptation exploits similarity between training and inference instances (Li et al., 2018), also in unsupervised scenarios (Farajian et al., 2017). These studies have even led to the creation of adaptive MT commercial toolkits (Federico, 2018). Importantly, in this task there is no target domain data available.

Unsupervised learning focuses on using monolingual corpora to train the translation system, without relying on any parallel corpora (Artetxe et al., 2018; Lample et al., 2018).

Active learning aims at selecting the most useful source sentences from a monolingual set and query their translation. This selection needs to minimize the post-edited cost and maximize the improvement of a finetuned model (Liu et al., 2018).

Interactive learning relies on a joint collaboration between a human and an MT system to obtain

Proceedings of the 5th Conference on Machine Translation (WMT), pages 56–64

high-quality translations while reducing the human effort in the process (Peris et al., 2016).

Our lifelong learning setting differs from both domain and instance-based adaptation, as it depends on the target data (called the lifelong data). The lifelong data set, unlike the training data, is unsupervised. Therefore, it is advisable to use techniques such as unsupervised learning, active learning, or interactive learning to approach the task.

3 Overview of the system and environment

The toolchain developed to evaluate the autonomous systems is described in Figure 1. It is made of four parts:

- the input datasets (purple on Figure 1), see section 3.1;
- the four blocks of the system (green on Figure 1 to be modified to include your own system), see appendix A for more details;
- the user simulation (orange on Figure 1), see section A.5;
- the evaluation blocks (blue on Figure 1), see section 3.2

Note that input datasets, user simulation and evaluation blocks are fixed and guarantee the reproducibility of the experiments. Participants are free to edit the four blocks of the system in order to include their own code. Once your code is included in this toolchain, the system will run automatically and the BEAT platform is responsible for managing the data exchanges between the different blocks of the architecture. Thus, you don't need to take care about the communication between blocks, especially, the interaction between the system and the user simulation is automatic.

3.1 Datasets

Two different datasets are available: the **training** data and what we called **lifelong** data. The **training** data is used to train the preprocessing system (eventually) and the initial system in a supervised way. Source text along with the translation of all documents included in this set are available at any time during the lifelong MT process. Note that no development data is provided, meaning that it is up to the participants to decide how to split the training data into train and development (if one is needed).

This year, we used the Europarl and NewsCommentary corpora as training data as they have document information along with their production dates. This represents between 50M and 58.6M words per language depending on the considered language pair (see details in Table 1).

The **lifelong** data is available in a sequential manner: each document is processed one after the other to simulate the process along time. This data is unsupervised, meaning that no reference translation is provided (they correspond to the data to translate every day). The system has to provide translations for those documents that will be evaluated.

We used the WMT14 English to French and English to German corpus as lifelong learning data. While this allows for comparison with systems that participated in WMT14 News translation shared task, one must keep in mind that the training data is much smaller than what was available for the shared task at the time. The aim here is to demonstrate the effectiveness of the continuous adaptation when compared to a baseline system that does not evolve (lower bound) and the best supervised system (retrained with all available data). In the future, we will extend the lifelong learning data to include that from 2014 up to the most recent one.

Training data (from 01.01.1996 to 31.12.2013)				
	English	French	English	German
#Documents	15218		15472	
#Segments	2308516		2246090	
#Words	55.6M	58.6M	53.6M	50.4M
Lifelong data (newstest2014)				
	English	French	English	German
#Documents	176		164	
#Segments	3003		3003	
#Words	62.3k	69.6k	59.3k	55.1k

Table 1: Statistics of the newstest2014 English-French and English-German corpora.

3.2 Evaluation

The evaluation is performed in the *mt_evaluation* and *BLEU_collate* blocks. The first block is aimed at collecting scoring statistics for the document being currently processed. In our case, it will correspond to the BLEU modified n-gram precisions. The second block will aggregate those statistics along with the penalisation in order to provide a final score for the system.

Each time the user simulation is asked for help,

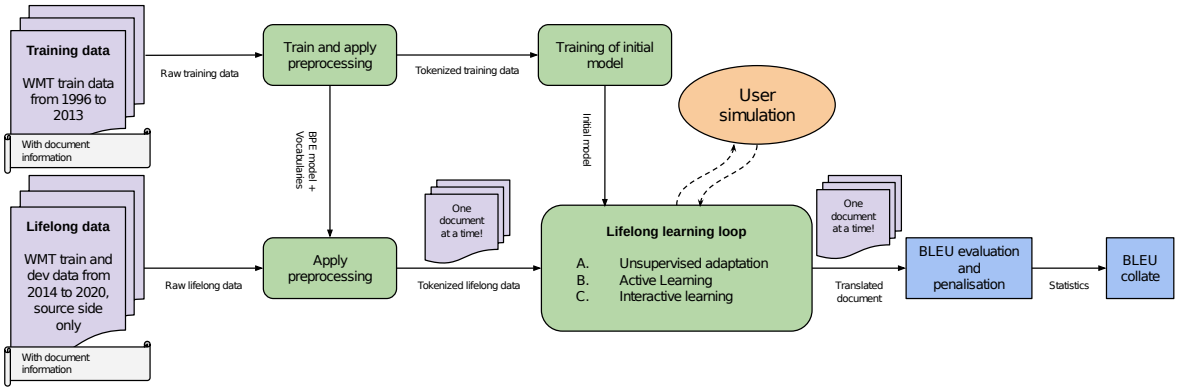

Figure 1: Flowchart of the lifelong learning MT system running on the BEAT platform.

a penalisation is calculated based o the request. The final penalised score S_{pen} corresponds to the following score:

$$S_{pen} = S_{adapt} + (S_{imp} - S_{cor})$$

with S_{adapt} being the score of the adapted system and S_{imp} and S_{cor} are the scores of this system where all sentences requested to the user simulation are considered entirely wrong and correct, respectively. Note that in the case of BLEU, the brevity penalty is not impacted by this calculation, only the correct n-gram counts will be decreased proportionally to the sentence requested for translation. For more details, see (Prokopalo et al., 2020).

4 Baseline systems

Integrating an NMT system in the BEAT platform requires to rethink the code so that everything is done in memory. We chose to use the nmtpytorch toolkit to implement the baseline systems (Caglayan et al., 2017).

Our baseline systems consists of a 2-layer bidirectionnal GRU (Cho et al., 2014) encoder and a 2-layer Conditional GRU decoder (Sennrich et al., 2017) equipped with an attention mechanism (Bahdanau et al., 2014) as implemented in nmtpytorch.

Given a source sequence of embeddings $X=\{x_1, \ldots, x_S\}$ and a target sequence of embeddings $Y=\{y_1, \ldots, y_T\}$, the bidirectional encoder first computes the sequence of annotations corresponding to the concatenation of the hidden states of the two GRU $A=\{a_1, \ldots, a_S\}$. At a given timestep t of decoding, the output layer estimates the probability of the next target word y_t as follows:

$$\begin{aligned} d_t &= \text{GRU}(y_{t-1}, d'_{t-1}) \\ c_t &= \text{Attention}(A, \text{query} \leftarrow d_t) \\ d'_t &= \text{GRU}'(c_t, d_t) \\ o_t &= \tanh(\mathbf{W_c} c_t + \mathbf{W_d} d'_t + \mathbf{W_y} y_{t-1}) \\ l_t &= \mathbf{W_o}(\mathbf{W_b} o_t + b_b) + b_o \\ P(y_t|X, Y_{<t}) &= \text{softmax}(l_t) \end{aligned} \quad (1)$$

For a single training sample, we then maximise the joint likelihood of source and target sentences:

$$\text{L}(X, Y) = \sum_{t=1}^{T} \log\left(P\left(y_t|X, Y_{<t}\right)\right) \quad (2)$$

5 Adaptation techniques

The first adaptation technique used is rather simple. It consists of selecting N sentences from training data that are the closest to the sentences in the document. The chosen similarity metric is the cosine between sentence embeddings obtained by a simple average of word embeddings, as described in (Arora et al., 2017). This data is then used to finetune the initial model for maximum 10 epochs with a learning rate of 0.00004, which is ten times smaller than during initial training of the model.

Furthermore, we employed an active learning strategy as an adaptation method. In principle, there are two steps involved. Firstly, the model provides a translation for each document from the lifelong learning corpus. As the lifelong learning data are unsupervised; therefore, a quality estimation (QE) technique is used to evaluate the quality of the translations without any access to a reference translation. Every document is evaluated

using sentence-level HTER scores (Specia et al., 2018). Secondly, an OpenKiwi QE model (Kepler et al., 2019) is used to rank the sentences according to their quality, and those with the worst HTER score are sent to the user simulation (active learning), which provides the correct translation of the selected sentences. This process implies the penalisation of the BLEU score as explained in Section 3.

6 Experimental setup

The dimensions of embeddings and GRU hidden states are set to 128 and 256, respectively. The embeddings are shared in the decoder (Press and Wolf, 2017). We use ADAM (Kingma and Ba, 2014) as the optimiser and set the learning rate and mini-batch size to 0.0004 and 64, respectively. Regularisation is done by means of a weight decay of $1e{-}5$ and the use of dropout on the embeddings, the source context and the output (set at 0.4) (Srivastava et al., 2014). We clip the gradients if the norm of the full parameter vector exceeds 1 (Pascanu et al., 2013).

The data is processed by a joint BPE model with 30k merge operations (Sennrich et al., 2016a). This leads to respectively 20.7k and 25.1k units for English and French and 17.2k and 26.5k units for English and German, respectively.

We train each model for a maximum of 100 epochs and early stop the training if validation BLEU (Papineni et al., 2002) does not improve for 10 epochs (Figure 2). We also halve the learning rate if no improvement is obtained for three epochs. The number of learnable parameters is around 8.7M for En-Fr and 8.5M for En-De.

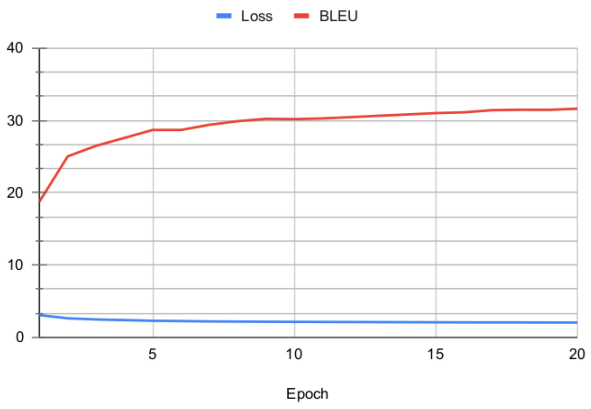

Figure 2: Training loss and BLEU scores for the English→German MT system.

7 Results

The results of the baseline systems and the adapted ones can be found in Table 2.

	English→French	English→German
Baseline		
SHEFFIELD	25.7	15.6
UPC	26.2	14.7
Data selection + finetuning		
SHEFFIELD	26.4	15.5
UPC	26.4	15.1

Table 2: BLEU scores on the newstest2014 English→French and English→German.

Results show that a simple data selection method along with finetuning can provide a small improvement of the system's performance for English to French. German is known to be a more complicated language, as demonstrated by the lower results and the inefficient effect of the adaptation method.

8 Discussion and next year evaluation

We can see that the task, given the very constrained data is very hard. A simple comparison with the results of the systems that participated in the WMT14 News translation task shows more than 10 BLEU points difference. We insist on the fact that the main goal of the challenge is to provide new methods to incrementally adapt the model to incoming documents. Without loss of generality, it is very probable that even with a better baseline system (trained on more data), the adapted models would exhibit a similar improvement.

Many questions and challenges remain open as to how lifelong learning for MT should be implemented. Next year, we ought to push further the evaluation by improving the QE model in order to better select the sentences to be sent to the user Simulation (Active Learning module). Hence, this will require to reconsider how the systems are evaluated. This year, we introduced a way of penalising the systems but without corresponding results.

We hope to have more participants bringing new ideas either by using the current baseline models (and avoiding the integration burden) or by integrating their own systems into the platform.

Acknowledgments

This task is organised by the University of Sheffield (Loic Barrault), University of Le Mans (Fethi Bougares), Universitat Politècnica de Catalunya

(Marta R. Costa-jussà and Magdalena Biesialska) and LNE (Olivier Galibert) in the framework of the EU Chist-ERA funded ALLIES project. This work is is supported in part by the Spanish Ministerio de Ciencia e Innovación, through the postdoctoral senior grant Ramón y Cajal and by the Agencia Estatal de Investigación through the projects EUR2019-103819, PCIN-2017-079 and PID2019-107579RB-I00 / AEI / 10.13039/501100011033.

References

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. 2017. A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations, ICLR 2017*.

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. 2018. Unsupervised neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, arXiv:1409.0473.

Magdalena Biesialska, Katarzyna Biesialska, and Marta R. Costa-jussà. 2020. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, Online. Association for Computational Linguistics.

Ozan Caglayan, Mercedes García-Martínez, Adrien Bardet, Walid Aransa, Fethi Bougares, and Loïc Barrault. 2017. Nmtpy: A flexible toolkit for advanced neural machine translation systems. *Prague Bull. Math. Linguistics*, 109:15–28.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

M. Amin Farajian, Marco Turchi, Matteo Negri, and Marcello Federico. 2017. Multi-domain neural machine translation through unsupervised adaptation. In *Proceedings of the Second Conference on Machine Translation, WMT 2017, Copenhagen, Denmark, September 7-8, 2017*, pages 127–137. Association for Computational Linguistics.

Marcello Federico. 2018. Challenges in adaptive neural machine translation. In *Proceedings of the AMTA 2018 Workshop on Translation Quality Estimation and Automatic Post-Editing*, pages 207–242, Boston, MA. Association for Machine Translation in the Americas.

Robert M. French. 1999. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128 – 135.

Fábio Kepler, Jonay Trénous, Marcos Treviso, Miguel Vera, and André F. T. Martins. 2019. OpenKiwi: An open source framework for quality estimation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics–System Demonstrations*, pages 117–122, Florence, Italy. Association for Computational Linguistics.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Philipp Koehn and Josh Schroeder. 2007. Experiments in domain adaptation for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, StatMT '07, page 224–227, USA. Association for Computational Linguistics.

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations*.

Xiaoqing Li, Jiajun Zhang, and Chengqing Zong. 2018. One sentence one model for neural machine translation. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*, Miyazaki, Japan. European Languages Resources Association (ELRA).

Ming Liu, Wray Buntine, and Gholamreza Haffari. 2018. Learning to actively learn neural machine translation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 334–344, Brussels, Belgium. Association for Computational Linguistics.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford neural machine translation systems for spoken language domain. In *International Workshop on Spoken Language Translation*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of*

the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54 – 71.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA. PMLR.

Álvaro Peris, Miguel Domingo, and Francisco Casacuberta. 2016. Interactive neural machine translation. *Computer Speech & Language*, 45.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163, Valencia, Spain. Association for Computational Linguistics.

Yevhenii Prokopalo, Sylvain Meignier, Olivier Galibert, Loïc Barrault, and Anthony Larcher. 2020. Evaluation of lifelong learning systems. In *Language Resources and Evaluation (LREC)*, Marseille, France.

Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nădejde. 2017. Nematus: a toolkit for neural machine translation. In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 65–68, Valencia, Spain. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Lucia Specia, Carolina Scarton, and Gustavo Henrique Paetzold. 2018. Quality estimation for machine translation. *Synthesis Lectures on Human Language Technologies*, 11(1):1–162.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

A LLMT system

This section describes the four different blocks that compose the LLMT system. The architecture of the system has been developed according to standard MT architectures. In order to facilitate the development of your system and to provide a baseline, a complete implementation of a LLMT system using nmtpytorch (Caglayan et al., 2017) is provided on the evaluation web page, see http://www.statmt.org/wmt20/lifelong-learning-task.html for more details.

General note: the prototypes of the **process** functions **must not** be changed!

A.1 Train and apply preprocessing

This block is responsible for preparing the training data. Preprocessing may include tokenization, learning subword decomposition model, etc. It is also responsible for creating the source and target vocabularies that will be used by the system. To do so, the entire training set is available at once (as in standard training protocol). The prepared training data is sent to the **train initial model** (sec. A.2) block while the subword model and vocabularies are sent to the **apply preprocessing** block (sec. A.3).

```
def process(self, data_loaders, outputs):
        # Get the training data
        data_loader = data_loaders[0]
        for i in range(data_loader.count()):
                (data, _, end_index) = data_loader[i]
                ... data["train_source_raw"].text
                ... data["train_target_raw"].text
                ... data["train_file_info"]
        #Note: setup_for_nmtpytorch(data_loaders) does that for you

        #HERE: DO AS MUCH DATA PREPARATION AS YOU WISH

        #Create vocabulary and BPE or SPM model
        data_dict_tok, src_vocab, trg_vocab, subword_model =
                preprocess(data_dict, self.source_language, self.target_language,
                        self.min_freq, self.short_list)

        data_dict_pickle = pickle.dumps(data_dict_tok).decode("latin1")

        #Write all the necessary outputs
        outputs['train_data_tokenized'].write({'text':data_dict_pickle}, end_index)
        outputs['source_vocabulary'].write({'text':src_vocab}, end_index)
        outputs['target_vocabulary'].write({'text':trg_vocab}, end_index)
        outputs['subword_model'].write({'text':subword_model}, end_index)

        # always return True, it signals BEAT to continue processing
        return True
```

A.2 Train initial model

The initial training of the system is implemented in the file *algorithms/loicbarrault/mt_train_model/1.py*. The process method is the main one. From this method, you can access all the training data from the **train_preprocessing** block. This block outputs a model.

```
# this will be called each time the sync'd input has more data available to be processed
def process(self, data_loaders, outputs):
        (data, _,end_data_index) = data_loaders[0][0]
        data_dict = pickle.loads(data["train_data"].text.encode("latin1"))

        #HERE: USE YOUR SOFTWARE FUNCTIONS TO TRAIN A MODEL

        # The model is Pickled with torch.save() and converted into a 1D-array of uint8
        # Pass the model to the next block
        outputs['model'].write({'value': model}, end_data_index)

        # always return True, it signals BEAT to continue processing
        return True
```

The data is available through the **data_loader**. In the provided baseline system, the processing consists of: tokenizing the data with Moses tokenizers (Koehn et al., 2007), training and applying a BPE model with subword_nmt (Sennrich et al., 2016b). As for the previous block, the **output** is written in the corresponding variable.

A.3 Apply preprocessing

The *apply preprocessing*'s algorithm is defined in the **process** function of the algorithm in *algorithms/loicbarrault/mt_apply_preprocessing/1.py* . The aim is to preprocess the lifelong data similarly to the training data using the vocabularies and subword models trained in the *train preprocessing* block.

The documents from the lifelong learning corpus are provided one after the other in the **input** parameter. Other information from previous blocks is available from the **data_loaders** as before.

```
# this will be called each time the sync'd input has more data available to be processed
def process(self, inputs, data_loaders, outputs):
  #Get the information from previous block,
  #NOTE: this should be done only once and stored in instance variable
  if self.src_bpe is None or self.trg_bpe is None \
        or self.src_vocab is None or self.trg_vocab is None:
    (data, _, end_data_index) = data_loaders[0][0]
    #Source and target vocabularies from the train_preprocessing block
    self.src_vocab = data["source_vocabulary"].text
    self.trg_vocab = data["target_vocabulary"].text
    #Source and target BPE objects to separate text into subwords units
    subword_model = io.StringIO(data["subword_model"].text)
    self.src_bpe = BPE(subword_model, vocab=self.src_vocab)
    self.trg_bpe = BPE(subword_model, vocab=self.trg_vocab)

  # Accessing lifelong data, one document at a time
  lifelong_source_raw = inputs['lifelong_source_raw'].data.text
  lifelong_target_raw = inputs['lifelong_target_raw'].data.text

  #HERE: APPLY THE PREPROCESSING TO THE DOCUMENT
  lifelong_source_tok = ...
  lifelong_target_tok = ...

  #Write all the necessary outputs
  outputs['lifelong_source_tokenized'].write({'text':lifelong_source_tok})
  outputs['lifelong_target_tokenized'].write({'text':lifelong_target_tok})
  if not inputs.hasMoreData():
    # DO SOMETHING WHEN ALL THE LIFELONG DATA HAS BEEN PROCESSED

  # always return True, it signals BEAT to continue processing
  return True
```

A.4 Lifelong learning loop

This block receives the initial model from the mt_train_initial_model block (sec. A.2) and process all files from the lifelong dataset provided by the *apply preprocessing* block, one at a time. This block has access to the whole training dataset and may store every processed document in memory in order to re-use it for further adaptation and/or any processing of your choice.

The output of this block is the translated document. This hypothesis might be obtained by simply translating the source document with the actual model (this is what the baseline model does). Eventually, you will plug your favorite unsupervised/semi-supervised or supervised adaptation scheme to create a better model before translating the document.

This module has also access to the user simulation (sec. A.5) from which the system can get reference translation for some segments in order to provide the best possible output.

A.5 User simulation

This module simulates the human in the loop. It receives requests from your system and provides answers to them. The requests and messages to the human are implemented in the *lifelong loop* block as dictionaries as follows:

```
request = {
  "request_type": "reference",
  "file_id": '{}'.format(file_id),
  "sentence_id": np.uint32(0)
}

message_to_user = {
  "file_id": file_id,  # ID of the file the question is related to
  "hypothesis": current_hypothesis[request['sentence_id']] ,
# The current hypothesis
  "system_request": request,  # the question for the human in the loop
}
```

As for now, only one type of request is available, namely 'reference'. This asks the user simulation to provide a correct translation for sentence number *sentence_id* from document *file_id*.

The answers are also a dict (see below) and can be obtained with the *validate* method as follows.

```
answer = {
  "answer": {"value": self.reference.text[sent_id]},
  "response_type": "reference",
  "file_id": self.file_info.file_id,
  "sentence_id": sent_id
 }
#Get the answer from the user simulation
human_assisted_learning, user_answer = loop_channel.validate(message_to_user)
```

Asking for human assistance is not free and will result in a penalisation of the system score, as described in sec. 3.2.

B How to setup a local platform for system development

B.1 Install

Installing the system requires to have a working conda[1] environment.

Then, the baseline system is available in the following repository: https://github.com/loicbarrault/allies_llmt_beat. Simply install using the *install.bash* script

B.2 Data

The data is available here: https://github.com/loicbarrault/allies_llmt_data. Simply follow the guidelines to recreate the data.

Update the *root_folder* at the bottom of the file *allies_llmt_beat/beat/databases/allies-mt-internal/1.json* with the path to the repository *allies_llmt_data/¡language-pair¿* directory (replace ¡language-pair¿ by the desired language pair, i.e. en-fr or en-de).

B.3 Run

Run the English→French system with the following command:

```
beat --prefix /path/to/git/allies_llmt_beat/beat exp run loicbarrault/loicbarrault/translation_ll_dev/1/translation_ll_dev
```

Run the English→German system with the following command:

```
beat --prefix /path/to/git/allies_llmt_beat/beat exp run loicbarrault/loicbarrault/translation_ll_dev/2/translation_ll_dev
```

[1] https://docs.conda.io/en/latest/