TransIns: Document Translation with Markup Reinsertion

Jörg Steffen
German Research Center for
Artificial Intelligence (DFKI)
Saarland Informatics Campus D3 2
66123 Saarbrücken, Germany
Joerg.Steffen@dfki.de

Josef van Genabith
DFKI and Saarland University
Saarland Informatics Campus D3 2
66123 Saarbrücken, Germany
Josef.Van_Genabith@dfki.de

Abstract

For many use cases, it is required that MT does not just translate raw text, but complex formatted documents (e.g. websites, slides, spreadsheets) and the result of the translation should reflect the formatting. This is challenging, as markup can be nested, apply to spans contiguous in source but non-contiguous in target etc. Here we present `TransIns`, a system for non-plain text document translation that builds on the Okapi framework and MT models trained with Marian NMT. We develop, implement and evaluate different strategies for reinserting markup into translated sentences using token alignments between source and target sentences. We propose a simple and effective strategy that compiles down all markup to single source tokens and transfers them to aligned target tokens. Our evaluation shows that this strategy yields highly accurate markup in the translated documents that outperforms the markup quality found in documents translated with popular translation services. We release `TransIns` under the MIT License as open-source software on https://github.com/DFKI-MLT/TransIns. An online demonstrator is available at https://transins.dfki.de.

1 Introduction

In MT research, models are usually trained and evaluated on plain text parallel data. But such models do not translate complex formatted documents created, e.g., with MS Office. Translating such documents comes with several challenges.

Text content has to be separated from formatting and other code and made available as input to MT. This requires a parser that handles the document format at hand and provides access to the embedded text content. After translation, the translated text must be placed into the target document and a further component is needed to create the translated version of the document reflecting the formatting and layout of the original document.

Furthermore, MT has to be able to handle *inline* sentence markup, i.e. to make sure that markup in the source sentence is correctly transferred to the appropriate parts of the target sentence. It is possible to train markup-aware MT models, e.g. by replacing tags with unique mask tokens in training and translation, as described in (Zhechev and van Genabith, 2010), but in order to use an existing MT model that is unaware of markup, the only option is to remove markup from the source sentence and to reinsert it at proper positions in the target sentence after translation. Du et al. (2010) describe a reinsertion strategy based on the phrase segmentation indicated by the decoder. This is refined by Hudik and Ruopp (2011) who use word alignments instead of phrase segmentation. Joanis et al. (2013) propose a hybrid approach combining phrase segmentation with word alignments. Building on these, Müller (2017) evaluates different markup handling strategies and provides implementations as part of the Zurich NLP `mtrain`[1] framework.

In order to utilize state-of-the-art MT technology and obtain alignments at the same time, we use Marian[2] NMT (Junczys-Dowmunt et al., 2018). Marian allows transformer models to be trained using guided alignment so that the decoder produces translations together with alignments between source and target tokens. The OPUS-MT[3] project (Tiedemann and Thottingal, 2020) provides pre-trained Marian models for many language pairs, mostly trained with guided alignment based on eflomal[4] word alignments (Östling and Tiedemann, 2016), but unaware of markup.

Below, we describe `TransIns`, a translator for non-plain text documents. We use the Okapi[5] framework to process such documents and extend

[1] https://github.com/ZurichNLP/mtrain
[2] https://marian-nmt.github.io/
[3] https://github.com/Helsinki-NLP/Opus-MT
[4] https://github.com/robertostling/eflomal
[5] https://okapiframework.org/

Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 28–34
November 7–11, 2021.

Okapi in order to query Marian for translations and alignments. We study different alignment-based markup reinsertion strategies, starting with the one implemented in `mtrain`. We identify deficits and present improved strategies. Finally, we evaluate different strategies and compare the markup quality between documents translated by `TransIns` and popular translation services.

2 Okapi Framework

Okapi is a free open-source framework designed to support localization and translation processes. It includes a collection of *filters* providing access to the translatable content for many file formats. A workflow in Okapi is modelled as a pipeline of *steps* that pass *events* through the pipeline. Events are associated with resources, e.g. text units, and are created when using a filter on a source document. A typical Okapi pipeline for translating documents consists of four steps:

The **Raw Document to Filter Events Step** reads the source document with an associated filter configuration and sends the filter events with the associated text units down the pipeline.

The **Segmentation Step** breaks down the text units into sentences, using rules specified in Segmentation Rules eXchange format (SRX)[6], a standard describing how to segment text.

The **Leveraging Step** sends each sentence to a translation service and stores the generated translation with the sentence. Translation services are accessed via *connectors*. Okapi provides connectors for popular translation services, but a connector for Marian is not included.

The **Filter Events to Raw Document Step** creates the target document in the original format from the translated text content coming in as filter events.

Okapi handles *global* document markup, but not *inline* sentence markup. This has to be dealt with by the translation service.[7]

3 TransIns System Description

Below, we describe how we build `TransIns` (**trans**lation with markup re**ins**ertion) based on the Okapi translation pipeline by adding Marian specific components and setting up an additional Okapi pipeline to support efficient processing.

Marian comes with a web-socket server that loads a model once at start time and then listens for single or batch translation queries. The server can be run remotely, supporting distributed setups with multiple Marian servers. In order to use Marian as a translation service from the leveraging step, we provide a Marian connector that implements the Okapi connector interface.

Most MT models are trained on parallel data where sentences are preprocessed, e.g. tokenized. Sentences to be translated need to be preprocessed in the same way. Also, the translation provided by the MT model might require postprocessing, e.g detokenization. With Marian, pre-/postprocessing often resorts to Perl scripts written for the Moses statistical MT system (Koehn et al., 2007). For `TransIns`, we use a Python reimplementation provided by Sacremoses[8]. Transformer MT models often apply subword tokenization in preprocessing. In postprocessing, subword tokenization has to be undone in the translated sentence. For transformer models, Byte-Pair Encoding (BPE) (Sennrich et al., 2016) and SentencePiece (Kudo and Richardson, 2018) are popular subword tokenizers. We use publicly available implementations of both subword tokenizers.[9,10] Undoing the subword tokenization in the translated sentence in postprocessing is straightforward by applying simple string replacements.

`TransIns` wraps the steps described above in a web service that provides corresponding endpoints for pre-/postprocessing single sentences or batches. The steps to apply can be configured separately for each translation direction. The Marian connector for Okapi calls this web service to preprocess a sentence before translation and again afterwards to postprocess the translated sentence.

The standard Okapi translation pipeline is somewhat inefficient: extracted sentences arrive at the leveraging step one-by-one, i.e. only single sentences are sent for pre-/postprocessing and translation to the corresponding services, even though the services support batch processing. The accumulated overhead of connecting and disconnecting slows throughput significantly. For efficient batch processing, we set up another Okapi pipeline, the *sentence collector pipeline*, consisting of a **Raw Document to Filter Events Step** and a **Segmentation Step** followed by a custom **Sentence Col-**

[6] https://www.gala-global.org/srx-20-april-7-2008

[7] The Okapi supported popular translation services can handle inline sentence markup, but details are not available.

[8] https://github.com/alvations/sacremoses

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

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

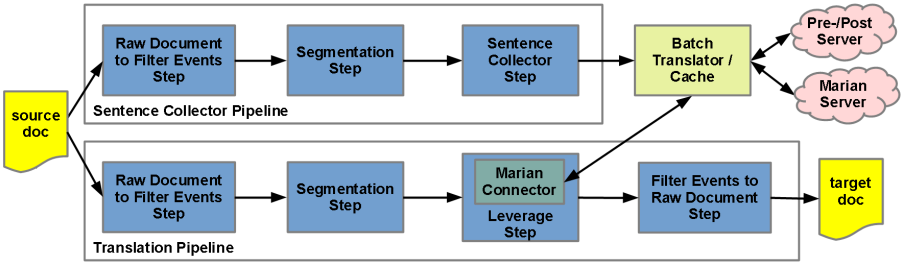


Figure 1: `TransIns` processes the source document with two Okapi pipelines

lector Step that simply adds each sentence to a local **Batch Translator** component. In a first run, we process the source document with this pipeline. Once the batch translator has collected all sentences it sends batches to preprocessing, translation and postprocessing. The batch translator also serves as **Cache**: for each source sentence, it stores the resulting target sentence. In a second run, we process the source document with the translation pipeline, but we adapt the Marian connector so that it queries the batch translator cache. This avoids connections to remote services, resulting in increased throughput. Figure 1 shows the `TransIns` workflow with both Okapi pipelines.[11]

4 Markup Reinsertion

Okapi provides sentences in a generic format. Sentence internal tags are encoded using characters from the Unicode private use area (PUA). Each tag consists of two such characters. The first encodes the type of the tag, which is either opening (e.g. `<b>`), closing (e.g. `</b>`), or isolated (e.g. `<br/>`). The second character encodes a running tag index. Tags provided by Okapi are always well-formed and balanced, i.e. for each opening tag, there is a corresponding closing tag, and tag pairs are properly nested. Tag indices are unique, and so is each tag pair.

The OPUS-MT models we use with `TransIns` are unaware of markup, so the only option is to remove tags before translation and to reinsert them afterwards. The general workflow of all `TransIns` markup reinsertion strategies is shown in Figure 2.

Alignments provided by Marian[12] refer to token indices. As tokenization is decided by preprocessing, we can only map and remove tags from the source sentence *after* preprocessing. Postprocessing like detokenization might change target tokenization. Therefore tag reinsertion has to be done

1 preprocess source sentence;
2 map each tag to a source token;[13]
3 remove tags from source sentence;
4 send source sentence to Marian for translation, retrieve target sentence and alignments;
5 reinsert tags into target sentence based on the alignments of the mapped source tokens;
6 clean up target sentence markup;
7 postprocess target sentence.

Figure 2: Markup reinsertion workflow

into the raw target sentence *before* postprocessing.

As reinserted tags may end up anywhere and in any order in the target sentence, we do a cleanup step after tag reinsertion that takes care of:

Tags in subword token sequences: If a tag is inserted within a sequence of subword tokens, e.g. as created by BPE, that tag has to be moved so that the subword token sequence can be properly merged in postprocessing. We move opening and isolated tags to the front of the sequence and closing tags to the end.

Improper tag order: Reinserted tags in the target sentence might occur in incorrect order, i.e. closing tag before corresponding opening tag. In this case, we swap the tags. Furthermore, tag pairs might be improperly nested. We reorder and, if required, insert additional tags to restore a proper nesting. E.g., a target sentence with two "overlapping" tag scopes

(1) x y z

is fixed by adding two additional tags

(2) x y **** **** z

Below, we describe the markup reinsertion strategies implemented in `TransIns`. For some strategies the cleanup step has to be adapted or extended.

4.1 `mtrain` Strategy

`mtrain` (Müller, 2017) is implemented in the Zurich NLP `mtrain` Python package.[14] We reimplement it with the `TransIns` workflow (Figure

[11] For debugging purposes, our system can be run without the sentence collector pipeline.

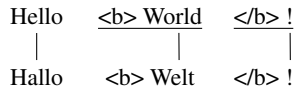
[12] using the `--alignment hard` option

[13] Source tokens include subword tokens.

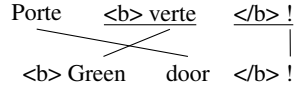
[14] https://github.com/ZurichNLP/mtrain/blob/master/mtrain/preprocessing/reinsertion.py#L315

2) as follows: in step 2, tags are always mapped to the *following* token.[15] In step 5, we iterate over the target tokens left to right. For each target token for which there is an alignment with a source token, we check if that source token has a tag mapped to it. If yes, that tag is *moved* in front of the target token. After the iteration, any remaining source sentence tags that have not been moved are added at the end of the target sentence. Such tags occur if they are mapped to a source token without alignment.

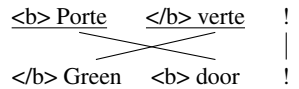
Example (3) demonstrates the strategy. The first row contains the source sentence with tags. Tag and mapped token are underlined. The last row contains the target sentence with reinserted tags, with vertical lines indicating alignments.

(3)


 is mapped to "World", is mapped to "!". "Welt" is aligned with "World", so is moved in front of "Welt". The same is done for "!" and , resulting in correct markup reinsertion. But if the word order in the target language is different from the source language, `mtrain` fails:

(4)


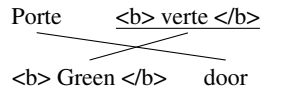
In (4) "door" is incorrectly rendered bold. In (5) both "Green" and "door" end up with incorrect markup:[16]

(5)


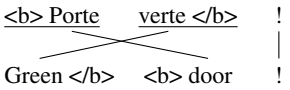
In conclusion, `mtrain` is only suitable for language pairs with similar word order and often fails otherwise. Below, we propose improvements to compensate for `mtrain` deficits.

4.2 `mtrain++` Strategy

The first improvement comes from the insight that tags do not generally refer to the *following* token. `mtrain++` only maps opening and isolated tags to the following token, but closing tags are mapped to the *previous* token and moved *after* the aligned target token. This fixes (4) (and also (5)):

(6)


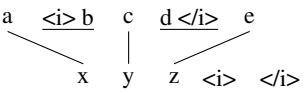
But `mtrain++` requires an adaptation of the cleanup step, as (7) shows:

(7)


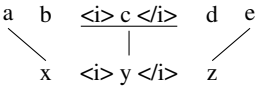
Swapping the tags here is not sufficient. We also have to consider the tag type. After swapping, we move the opening tag in front of the previous token and the closing tag after the following token:

(8) Green door !

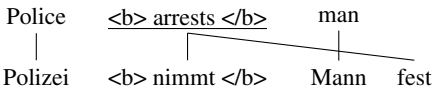
Another deficit of `mtrain` is the handling of tags mapped to unaligned source tokens.[17] Unaligned tags are added at the end of the target sentence, resulting in a counter-intuitive markup reinsertion:

(9)


`mtrain++` handles unaligned tags in the source sentence: opening and isolated unaligned tags are moved *in front of* the following aligned token and closing unaligned tags *after* the previous aligned token. We then remap the tags accordingly. This results in correct markup reinsertion:

(10)


A further problem may occur with 1-to-n alignments where a single source token is aligned to multiple target tokens. Tags mapped to such a source token are *moved* by `mtrain`, so they are only applied to the first of the possible n aligned target tokens:

(11)


`mtrain++` *copies* opening and closing tags instead of moving them, resulting in correct markup reinsertion for (11), where both "nimmt" and "fest" are correctly rendered bold.

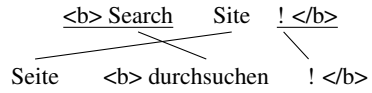
But copying tags instead of moving them comes at a price: tags in the target sentence are potentially no longer well-formed and balanced. E.g., an opening tag may be copied twice while the corresponding closing tag is only copied once if they are mapped to different source tokens. We extend the cleanup step so that well-formedness and balance are restored.

But even with all the improvements, there still remain configurations where the markup is not reinserted correctly to the target sentence:

[15] A tag at the end of a sentence is mapped to an artificial end-of-sentence token.

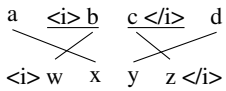
[16] and are swapped in the cleanup step.

[17] We call such tags *unaligned tags* in the following.

(12)


The underlying problem is that `mtrain++` only considers tokens immediately next to tags. Source tokens located within a tag pair's scope but not next to a tag, as "Site" in (12), may be aligned with a target token outside of that tag pair's scope in the target sentence.

Another problem are tag pairs with a contiguous scope in the source sentence that should be non-contiguous in the target sentence:

(13)


In (13) only w and z should be rendered italic.

Rather than continuing to fix individual problem cases, below we develop a new mapping strategy designed to provide a general solution.

4.3 Complete Mapping Strategy (CMS)

CMS is based on the insight that if (i) alignments steer markup transfer, and (ii) alignments relate tokens, then compiling down tag pairs to their minimal token level scope should solve most if not all of the problems presented in Sections 4.1 and 4.2. Formally, a tagged sentence is a sequence $s = s_1 \dots s_m$ consisting of one or more sequence elements s_i, where s_i is either a raw token w or a function $t_i(s_j)$ representing a tag pair with scope over a sequence s_j.[18] This allows for sentences with arbitrarily nested tag pairs and ensures that markup is well-formed. We use $t_i^1(\dots(t_j^k(.)))$ to denote one or more (k) nested tag pairs with scope $(.)$. The algorithm compiles tag pairs to their minimal scopes $t_i^1(\dots(t_j^k(w)))$ by repeatedly applying

(14) $t_j^k(s_1 \dots s_m) \rightarrow t_j^k(s_1) \dots t_j^k(s_m)$

until no tag pair with scope sequence of length > 1 remains. In terms of tag mapping, this results in each tag pair being mapped to *all* tokens in its original scope, hence the term *complete* mapping.[19] The compilation is meaning-preserving, e.g.

(15) x y z

is turned into

(16) x   y   z

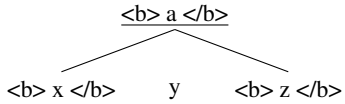
CMS simplifies markup reinsertion significantly: as minimum scope tag pairs "travel" with token alignments, tag balance, well-formedness and complex positioning in the target sentence are taken care of by the alignments. Tag swapping, reordering or insertion is no longer required in the cleanup step. Unaligned opening and closing tags no longer have to be moved.[20] The cleanup step only needs to take care of tags in subword token sequences. Readers are invited to check that all "toxic" examples discussed in Sections 4.1 and 4.2 are handled correctly by CMS. Finally to remove clutter, we simplify the target sentence markup by eliminating closing tags immediately followed by the corresponding opening tag, so

(17) x ** ** y ** ** z

simplifies to

(18) x y z

While CMS ensures that target tokens "inherit" the markup of their aligned source tokens, there may be unaligned target tokens. Such tokens would never receive markup, resulting in *gaps*:

(19)


In such configurations, rendering "y" bold seems appropriate. We implement a *tag interpolation* scheme for target tokens within gaps. A gap is a target token sequence not longer than a specified maximum gap size where all target tokens have no markup, either because they are unaligned or they are aligned with a source token without tags. The latter is often the result of incorrect alignments that tag interpolation can correct.[21] Tag interpolation inspects the tags applied to the neighbor tokens in front of and after the gap. Identical tags[22] found with both neighbor tokens are applied to the gap tokens. E.g., applying tag interpolation to a gap of size 2 with tokens x and y turns

(20)   w   x y z

(after simplification) into

(21)   w x y z

5 Evaluation

To the best of our knowledge, there are no public standard evaluation data sets for markup transfer yet. We collect 10 complex web pages from spiegel.de, a well-known German news provider, that contain a total of 378 sentences with inline markup. We convert the web pages to MS Office

[18] Isolated tags are considered as tag pairs with empty scope, though they have an implicit scope over all following tokens.

[19] Isolated tags are still mapped to the following token.

[20] Unaligned isolated tags are still moved in front of the following aligned token.

[21] Tag interpolation produces a markup error if a gap target token is correctly aligned with a source token without tags.

[22] Identical tags have the same tag type and index.

Reinsertion Strategy		de –> fr		de –> en	
		Marian Alignments	Perfect Alignments	Marian Alignments	Perfect Alignments
mtrain		104	94	98	85
mtrain++		23	31	28	31
CMS	max gap 3	12 (8 + 4)	4	7 (3 + 4)	4
	max gap 2	9 (8 + 1)	1	7 (3 + 4)	4
	max gap 1	**9 (8 + 1)**	1	**5 (3 + 2)**	2
	max gap 0	10 (10 + 0)	0	**5 (5 + 0)**	0

Table 1: Markup transfer errors by reinsertion strategy

MT Service	de –> fr	de –> en
Google	119	55
DeepL	129	78
Microsoft	261	235
CMS (max gap 1)	58 (48 + 10)	32 (21 + 11)
CMS (max gap 0)	**57 (57 + 0)**	**25 (25 + 0)**

Table 2: Markup transfer errors by translation service

docx documents, preserving the relevant markup, and examine translations to French and English using the latest OPUS-MT models. As a quality measure of markup transfer, we count how many target tokens end up with incorrect markup.[23]

Even though CMS can handle all toxic examples described in Section 4, as a sanity check, we do a small evaluation of the three `TransIns` reinsertion strategies using the first evaluation document containing 40 sentences with inline markup. We do this for alignments as provided by the OPUS-MT models and also for hand-corrected perfect alignments.[24] For CMS and Marian alignments, we distinguish between errors resulting from incorrect alignments (first number in brackets) and errors from incorrect tag interpolation (second number in brackets). We also examine different maximum gap sizes. Table 1 shows the results.

For both translation directions `mtrain` produces most errors. Using perfect alignments yields only a minor improvement. `mtrain++` reduces the number of errors by about two-thirds. Surprisingly, the quality for both translation directions is slightly better when using Marian alignments instead of perfect alignments. This is due to the fact that a single change in alignment, even if it is a correction, can potentially change the markup of multiple tokens. With CMS, a change in alignment only effects a single target token, making this strategy less volatile.

For both types of alignments, CMS produces the smallest number of errors. The results are similar for both translation directions. For de –> fr, using Marian alignments with a maximum gap size of 3, we find 8 errors caused by incorrect alignments and 4 errors caused by incorrect tag interpolation. Reducing the maximum gap size to 2 and 1 decreases the number of interpolation errors, as tag interpolation is applied to fewer gaps. With a gap size of 0, i.e. with no tag interpolation, the number of errors caused by incorrect alignments increases by 2. These errors are now no longer corrected by tag interpolation. With perfect alignments, only the errors caused by incorrect tag interpolation remain.

The main focus of our evaluation is the comparison of CMS with popular translation services. These services are able to handle markup, but the details are unknown to us. Table 2 shows the errors for all evaluation document translations.[25] The performance for de –> en is always better than for de –> fr. This is probably due to the more similar word order between German and English. For both translation directions, CMS produces less than half as many markup errors as the next best commercial MT service. Errors decrease when omitting tag interpolation, i.e. the number of corrected alignment errors is smaller than the number of errors introduced by incorrect tag interpolation. We see this as an indicator of the high quality alignments provided by the OPUS-MT models.

6 Conclusion

In this paper, we present `TransIns`, an open-source system implementing several alignment based strategies for markup reinsertion in translated documents. `mtrain` constitutes a baseline, while `mtrain++` can handle more complex configurations. CMS correctly handles all problem cases discussed and outperforms the markup transfer in documents translated with popular translation services.

[23] We ignore punctuation tokens with incorrect markup.

[24] We hand-correct 8% of the de –> fr and 10% of the de –> en Marian token alignments.

[25] All evaluation documents are available at https://github.com/DFKI-MLT/TransIns/tree/master/evaluation.

Acknowledgements

We thank Thierry Declerck and Cristina España i Bonet for feedback and productive discussion. We also thank the anonymous reviewers for their constructive reviews. This work is supported in part by the German Federal Ministry of Education and Research (BMBF) under funding code 01IW20010 (CORA4NLP).

References

Jinhua Du, Johann Roturier, and Andy Way. 2010. TMX markup: A challenge when adapting SMT to the localisation environment. In *Proceedings of the 14th Annual conference of the European Association for Machine Translation*, Saint Raphaël, France. European Association for Machine Translation.

Tomáš Hudik and Achim Ruopp. 2011. The integration of Moses into localization industry. In *Proceedings of the 15th Annual conference of the European Association for Machine Translation*, Leuven, Belgium. European Association for Machine Translation.

Eric Joanis, Darlene Stewart, and Samuel Larkin. 2013. Transferring markup tags in statistical machine translation: A two-stream approach. In *Proceedings of MT Summit XIV Workshop on Post-editing Technology and Practice*, pages 73–81.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, André F. T. Martins, and Alexandra Birch. 2018. Marian: Fast neural machine translation in C++. In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.

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.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Mathias Müller. 2017. Treatment of markup in statistical machine translation. In *Proceedings of the Third Workshop on Discourse in Machine Translation*, pages 36–46, Copenhagen, Denmark. Association for Computational Linguistics.

Robert Östling and Jörg Tiedemann. 2016. Efficient word alignment with Markov Chain Monte Carlo. *Prague Bulletin of Mathematical Linguistics*, 106:125–146.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. 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.

Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT – building open translation services for the world. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 479–480, Lisboa, Portugal. European Association for Machine Translation.

Ventsislav Zhechev and Josef van Genabith. 2010. Seeding statistical machine translation with translation memory output through tree-based structural alignment. In *Proceedings of the 4th Workshop on Syntax and Structure in Statistical Translation*, pages 43–51, Beijing, China. Coling 2010 Organizing Committee.