

PyMarian: Fast Neural Machine Translation and Evaluation in Python

Thamme Gowda¹ Roman Grundkiewicz¹ Elijah Rippeth²
Matt Post¹ Marcin Junczys-Dowmunt¹

¹ Microsoft Translator

{thammegowda, rogrundk, mattpost, marcinjd}@microsoft.com

² University of Maryland

erip@cs.umd.edu

Abstract

The deep learning language of choice these days is Python; measured by factors such as available libraries and technical support, it is hard to beat. At the same time, software written in lower-level programming languages like C++ retain advantages in speed. We describe a Python interface to Marian NMT, a C++-based training and inference toolkit for sequence-to-sequence models, focusing on machine translation. This interface enables models trained with Marian to be connected to the rich, wide range of tools available in Python. A highlight of the interface is the ability to compute state-of-the-art MT evaluation metrics, such as COMET and BLEURT, from Python but using Marian’s inference engine, with a speedup factor of up to $7.8\times$ the existing implementations. We also briefly spotlight a number of other integrations, including Jupyter notebooks, connection with prebuilt models, and a web app interface provided with the package. PyMarian is available in PyPI via `pip install pymarian`.

1 Introduction

Marian NMT¹ (Junczys-Dowmunt et al., 2018a) was one of the earliest training and inference toolkits for sequence-to-sequence-based machine translation. Originally written under the name `amun` and providing fast inference for Groundhog-trained models,² it was quickly built up to also provide speedy, reliable multi-GPU and multi-node training of Transformer models, along with many other features. It has been widely used in commercial production settings (Junczys-Dowmunt et al., 2018b), for academic and industrial research, for the distribution of pre-trained models (Tiedemann and Thottingal, 2020a), and as the basis for extremely fast in-browser translation (Bogoychev et al., 2021).

Many of these features were enabled by its efficient C++-backend, but it must be admitted that this

dependency is also a barrier to many researchers, who increasingly work with Python. This paper describes a new set of Python bindings that have been added to Marian. Written using Pybind11, these bindings are available as a pip-installable Python package via the Python Package Index³ or can be installed from Marian’s source. We describe several features and applications facilitated by PyMarian:

- *Inference and training* (§ 2). It is easy to load Marian-trained models and send data through them for translation. This also makes it easy to translate with publicly-available models, and to plug them into other Python codebases.
- *Fast evaluation* (§ 3). Model-based metrics such as COMET and BLEURT have demonstrated their superiority, but their provided toolsets make them slow to compute. We provide `pymarian-eval`, which makes use of converted models, packaged in a Python CLI interface.
- *Example applications* (§ 4). We demonstrate the versatility of `pymarian` with a number of examples including a web-based demonstration framework.

A particular focus of the paper is in benchmarking popular COMET models reimplemented in Marian and available through PyMarian (§ 3.3), which run significantly faster than in their native implementations, providing up to $7.8\times$ speedup in a multi-GPU setting.

2 PyMarian API

PyMarian offers `pymarian` Python package containing convenient high level APIs. We use Pybind11⁴ to bind the Python calls to Marian C++

¹<https://marian-nmt.github.io>

²<https://github.com/pascanur/GroundHog>

³<https://pypi.org/project/pymarian>

⁴<https://github.com/pybind/pybind11>

APIs. PyMarian uses the same configuration system as Marian, however makes it Pythonic by offering keyword-argument (i.e., `**kwargs`).

At the package's top level, we have three classes: `Translator`, `Trainer` and `Evaluator`. First two are described in this section, while the evaluator is presented in details later in Section 3.

2.1 Translator

The Python API for decoding Marian models with beam search is provided by `Translator` class.

```
from pymarian import Translator
mt = Translator(
    models="model.ende.npz",
    vocabs=["vocab.spm", "vocab.spm"]
)
hyp = mt.translate("Hello world!")
print(hyp) # "Hallo Welt!"
```

It offers the same hyperparameters and functionalities as the translation service in C++, such as:

- Translation speed optimization with custom beam search sizes (`beam_size`), batch organization (`mini_batch`, `mini_batch_sort`), and `fp16`;
- n -best lists translation (`n_best=True`);
- Word alignments (e.g., `alignment="hard"`) and word-level scores (`word_scores=True`) when more detailed subword-level information is needed (`no_spm_encode=True`);
- Noised sampling from full distribution and top-K sampling with custom temperatures (e.g., `output_sampling="topk 100 0.1"`);
- Force-decoding of given target language prefixes (`force_decode=True`).

2.2 Trainer

Python API for training models supported in Marian toolkit is provided by the `Trainer` class.

```
from pymarian import Trainer
args = {
    "type": "transformer",
    "model": "model.npz",
    "train_sets": ["train.en", "train.de"],
    "vocabs": ["vocab.spm", "vocab.spm"],
}
trainer = Trainer(**args)
trainer.train()
```

Complete examples are available in Marian's source code in `src/python/tests/regression`.

3 Fast MT Evaluation in PyMarian

The Marian NMT had been a toolkit for translation and language modeling with the emphasis on speed. With the recent revision of Marian toolkit, we have implemented evaluation metrics, for both training and fast inferencing, while retaining its emphasis on speed. In addition, we have also enabled evaluator APIs in Python module, via a class named `Evaluator`.

3.1 Evaluator

`Evaluator` supports scoring MT hypothesis with either source, or reference, or both. Generally, evaluators are classified into reference-free (quality estimation) and reference-based types. We provide implementations of both types.

```
from pathlib import Path
from pymarian import Evaluator

evaluator = Evaluator.new(
    model_file="marian.model.bin",
    vocab_file="vocab.spm",
    like="comet-qe", quiet=True,
    fp16=False, cpu_threads=4)

srcs = ['Hello', 'Howdy']
mts = ['Howdy', 'Hello']
lines = (f'{s}\t{t}'
         for s,t in zip(srcs, mts))
scores = evaluator.evaluate(lines)
for score in scores:
    print(f'{score:.4f}')
```

3.2 Metrics

Along with providing implementation for the evaluator framework, we also provide checkpoints for some of the popular MT metrics, such as COMETs and BLEURT. Since the checkpoint file format of the existing metrics are incompatible with Marian toolkit, we have converted them to the required format and released on Huggingface.⁵ Table 1 shows the available models and their IDs on HuggingFace hub.

Using the `Evaluator` API, we have developed a convenient command-line utility named `pymarian-eval`, which internally takes care of

⁵<https://huggingface.co/models>

Metric	Fields	Reference	HuggingFace ID
bleurt-20	T, R	Sellam et al. (2020)	marian-nmt/bleurt-20
wmt20-comet-da	S, T, R	Rei et al. (2020b)	unbabel/wmt20-comet-da-marian
wmt20-comet-qe-da	S, T	"	unbabel/wmt20-comet-qe-da-marian
wmt20-comet-qe-da-v2	S, T	"	unbabel/wmt20-comet-qe-da-v2-marian
wmt21-comet-da	S, T, R	Rei et al. (2021)	unbabel/wmt21-comet-da-marian
wmt21-comet-qe-da	S, T	"	unbabel/wmt21-comet-qe-da-marian
wmt21-comet-qe-mqm	S, T	"	unbabel/wmt21-comet-qe-mqm-marian
wmt22-comet-da	S, T, R	Rei et al. (2022a)	unbabel/wmt22-comet-da-marian
wmt22-cometkiwi-da	S, T	Rei et al. (2022b)	unbabel/wmt22-cometkiwi-da-marian
wmt23-cometkiwi-da-xl	S, T	Rei et al. (2023)	unbabel/wmt23-cometkiwi-da-xl-marian
wmt23-cometkiwi-da-xxl	S, T	"	unbabel/wmt23-cometkiwi-da-xxl-marian
cometoid22-wmt21	S, T	Gowda et al. (2023)	marian-nmt/cometoid22-wmt21
cometoid22-wmt22	S, T	"	marian-nmt/cometoid22-wmt22
cometoid22-wmt23	S, T	"	marian-nmt/cometoid22-wmt23
chrfoid-wmt23	S, T	"	marian-nmt/chrfoid-wmt23

Table 1: List of metrics supported in pymarian, their required fields, reference, and HuggingFace model IDs. Fields S, T, and R are *source*, *translation* (also variously called the *candidate* or *hypothesis*), and *reference*, respectively.

downloading models from HuggingFace model hub and caching them locally.

We provide `-a|--average` option for obtaining the system level score only (`-a only`), segment level scores only (`-a skip`), or both where the average is appended (`-a append`). For example,

```
pymarian-eval -m wmt22-cometkiwi-da \
-s src.txt -t mt.txt -a only
```

The current toolkits that originally implement the popular metrics consume higher memory and time for loading the checkpoints than necessary. This is increasingly problematic as metric checkpoint files are getting bigger over the years. The format used by Marian is optimized for faster loading with minimal memory overhead. We present the model loading time and memory utilization in Table 2. For instance, consider `wmt23-cometkiwi-da-xl`, whose checkpoint file is 13.9GB.⁶ The *original* tool (`comet-score`) takes 27GB RAM and 530 seconds to warmup on 8 GPUs, where as `pymarian-eval` achieves the same in half the RAM and only 12 seconds.

3.3 Benchmarks

A concern with new implementations is the risk of producing incompatible results. We therefore compare our model conversion and implementations carefully so as to ensure that `pymarian-eval` produces the same results.

Our benchmark setup is as follows:

⁶`wmt23-cometkiwi-da-xxl` is 42.9GB and we were unable to load it on the GPUs used for benchmarks in this paper (32GB V100).

- Dataset: WMT23 General Translation submissions (Kocmi et al., 2023); we combine all systems for all languages pairs, which results in a total of 364,200 examples.
- COMET’s original implementation: `unbabel-comet v2.2.2` (Rei et al., 2020a); transitive dependencies: `torch v2.4.0`, `pytorch-lightning v2.3.3`, `transformers v4.43.3`
- BLEURT original implementation is installed from source repository;⁷ transitive dependencies: `tensorflow v2.17.0`
- Marian v1.12.31, compiled with GCC v11.
- Python v3.10.12, Ubuntu 22.04.3, on Intel(R) Xeon(R) Platinum 8168 CPU @ 2.70GHz
- GPU: 8x Nvidia Tesla V100 (32GB); Driver v525.105.17, CUDA v12.3
- Batch size is 128, except for `wmt23-cometkiwi-xl`, the largest batch size that worked are: 64 for eight GPUs and 32 for one GPU.

In Table 3, we report the time taken by original toolkits (Pytorch based `comet-score` and Tensorflow based `bleurt`) and our implementation. For ours, we report Marian (binary produced by C++), and `pymarian-eval` (with `float32` and `float16` precisions). In addition, we also present the average of segment scores, and error, i.e., the absolute difference between the scores produced by the

⁷<https://github.com/google-research/bleurt/tree/cebe7e6f>

Model	Time (seconds)						Memory (MB)			
	1 GPU			8 GPUs			1 GPU		8 GPUs	
	Orig	Ours	Speedup	Orig	Ours	Speedup	Orig	Ours	Orig	Ours
bleurt-20	23.7	3.0	7.9x	NA	8.4	NA	6,606	2,640	NA	3,455
wmt20-comet-da	37.0	4.6	8.0x	193.8	9.7	19.9x	5,387	2,782	5,388	3,598
wmt20-comet-qe-da	32.6	3.8	8.6x	197.3	8.9	22.1x	5,276	2,682	5,278	3,499
wmt22-comet-da	37.9	4.5	8.5x	193.5	9.7	20.0x	5,365	2,786	5,364	3,603
wmt22-cometkiwi-da	33.9	3.3	10.2x	199.1	8.8	22.7x	5,244	2,623	5,246	3,438
wmt23-cometkiwi-da-xl	108.5	7.5	14.4x	530.2	12.1	43.9x	27,554	13,815	27,554	14,631

Table 2: Model load time (seconds) and memory (megabytes) taken to initialize the models and score a single example. Marian and pymarian use memory-mapped files, which enable faster loading than original implementation. Numbers are the average of three runs.

Metric	Original	Time (seconds)			Speedup		
		Marian	PyM	PyM FP16	Marian	PyM	PyM FP16
<i>1 GPU</i>							
bleurt-20	2312±2.2	635±0.3	656±0.3	467±0.6	3.6x	3.5x	4.9x
wmt20-comet-da	3988±0.8	954±1.0	968±4.7	783±5.1	4.2x	4.1x	5.1x
wmt20-comet-qe-da	2529±0.4	608±3.7	623±3.6	501±0.3	4.2x	4.1x	5.0x
wmt22-comet-da	3772±1.3	858±4.6	884±4.5	676±0.8	4.4x	4.3x	5.6x
wmt22-cometkiwi-da	2357±2.0	419±0.4	437±1.7	327±1.0	5.6x	5.4x	7.2x
wmt23-cometkiwi-da-xl	17252±0.7	3405±4.7	3480±3.9	1949±3.1	5.1x	5.0x	8.8x
<i>8 GPUs</i>							
bleurt-20	NA	85±0.1	99±0.1	76±0.4	NA	NA	NA
wmt20-comet-da	926±1.0	125±0.1	146±0.7	124±1.0	7.4x	6.3x	7.5x
wmt20-comet-qe-da	622±0.1	82±0.1	95±0.2	81±0.2	7.6x	6.5x	7.7x
wmt22-comet-da	896±0.8	114±0.1	135±0.3	111±0.7	7.8x	6.6x	8.1x
wmt22-cometkiwi-da	562±0.7	59±0.1	72±0.1	58±0.1	9.5x	7.8x	9.6x
wmt23-cometkiwi-da-xl	3288±1.8	662±2.6	862±13.3	258±0.7	5.0x	3.8x	12.7x

Table 3: Time taken (seconds) to score the benchmark datasets having 364,200 examples, and the speedup of our implementation with respect to the original. Numbers are the average of three runs on one and eight GPUs. PyM is short for PyMarian. The column with FP16 is half-precision, and the rest are full-precision (32-bit).

original and ours. The scores and derived errors for our implementation remain consistent regardless of whether the C++ implementation is invoked via the command line binary (marian evaluate) or through the Python bindings wrapper (pymarian-eval). Additionally, the scores are identical whether the benchmarks are conducted on a single GPU or parallelized across multiple GPUs. We avoid repetition, and instead present only the values for full-precision (FP32) and half-precision (FP16). As shown in Table 4, ours yield the same scores as the original, with minor discrepancies attributable to floating-point calculations.

In addition to providing significantly faster processing times, pymarian-eval provides a flexible CLI tool with a natural POSIX interface (e.g., STDIN/STDOUT, use of TSV formats). This allows it to integrate well with other tools, such as SacreBLEU’s testset downloading capabili-

ties (Post, 2018).

4 Example applications

A Python API makes it simple to incorporate Marian models into the many Python-native settings that researchers are accustomed to. In this section we illustrate example use cases and applications of PyMarian, demonstrating its versatility.

4.1 Jupyter notebook

PyMarian makes it easy to use Marian-trained models in interactive sessions such as Jupyter Notebook-like⁸ environments. We provide an example notebook for translation, training, and evaluation via Google Colab at https://colab.research.google.com/drive/1Lg_W5K2nLtvaKfLuHjc-LAajenI_SGL3

⁸<https://jupyter.org>

Metric	Score			Error	
	Original	Marian FP32	Marian FP16	Marian FP32	Marian FP16
bleurt-20	0.7255	0.7252	0.7211	0.0003	0.0044
wmt20-comet-da	0.5721	0.5720	0.5716	0.0001	0.0005
wmt20-comet-qe-da	0.1933	0.1932	0.1924	0.0001	0.0009
wmt22-comet-da	0.8462	0.8461	0.8427	0.0000	0.0034
wmt22-cometkiwi-da	0.7984	0.7984	0.7981	0.0000	0.0003
wmt23-cometkiwi-da-xl	0.6840	0.6839	0.6862	0.0001	0.0023

Table 4: The average scores produced by the original implementation and ours. The columns named ‘Error’ are the absolute difference between the average of scores from the original and our implementations.

4.2 OPUS-MT models

Over the years, Marian NMT has been widely adopted by the community to train and release open-sourced machine translation systems. One of the largest projects developing such resources is OPUS-MT, which offers over 1,000 pre-trained models (Tiedemann and Thottingal, 2020b; Tiedemann et al., 2023). PyMarian provides a seamless interface to decode with these existing Marian-trained models.

4.3 Web-based interface

PyMarian permits easy connection from Marian models to Python’s visualization libraries. We incorporate a Flask-based web server that can display a range of models side by side.⁹ It supports loading of models from local disk (type “base”) or connecting to Microsoft’s API (type “mtapi”).

translators:

en-de-research:

```

type: base
name: research
model: /path/to/marian.npz
vocab: /path/to/vocab.spm

```

en-de-prod:

```

type: mtapi
name: prod
subscription-key: {redacted}
source-language: en
target-language: de

```

Figure 1 provides an example of this interface. Due to the flexibility of Python, extending the model to support other types is simple.

5 Related Work

A wide range of Python toolkits exist for training and inference for the “classical” (i.e., not LLM-based) sequence-to-sequence approach to machine

translation. One of the most popular is Meta’s fairseq (Ott et al., 2019), which supports a wide range of training and inference features, including multi-GPU and multi-node training. Amazon’s Sockeye (Hieber et al., 2022) is another option; while it has fewer features than fairseq, it is known for its strong software engineering practices and flexibility. Both of these toolkits are based on Pytorch (Paszke et al., 2019), and support research and production use cases. Sockeye has recently (as of June 7, 2024) been end-of-lived.¹⁰

A significant amount of research and development activity takes place using HuggingFace’s popular transformers package (Wolf et al., 2020). Work in this area tends to be much more research-focused, however, which means that software-engineering practices and speed are sacrificed in favor of rapid development. HuggingFace also provides a data store for a huge range of datasets and models. VLLM is a recent project that provides fast, production-oriented inference for HuggingFace models (Kwon et al., 2023). However, at the time of writing, VLLM primarily focused on decoder-only language models; it lacked support for MT evaluation metrics like BLEURT and COMETs, and encoder-decoder NMT models. Consequently, this difference has hindered direct comparison with our work. There is support for loading Marian models in HuggingFace transformers, largely provided by Tiedemann and Thottingal (2020a). However, not all Marian model features are supported. pymarian provides Python-based access to any Marian model, with C++ inference speeds.

Although pymarian does not aim to enhance the efficiency of Marian, it ensures that any impact on processing speed remains minimal while invoking highly efficient Marian C++ codebase from Python (see Table 3). Comparison of C++ Mar-

⁹<https://github.com/marian-nmt/pymarian-webapp>

¹⁰<https://github.com/aws-labs/sockeye/commit/4e2fbb30be9bca1f5073f092b687966636370092>

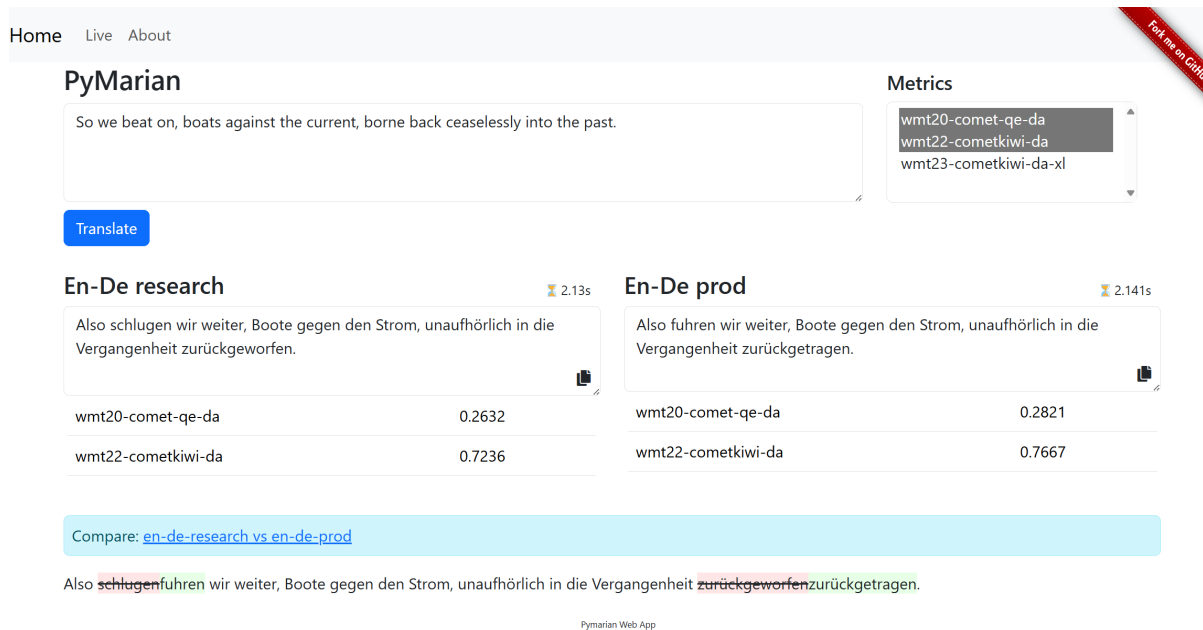


Figure 1: PyMarian web demo with two outputs, the diff between them, and a set of chosen quality-estimation metrics.

ian Translator with other NMT toolkits are in prior works that evaluate efficiency across frameworks, for both training (Wang et al., 2018) and inference.¹¹ Additionally, shared tasks that emphasize MT efficiency (Heafield et al., 2020, 2021, 2022) also offer valuable insights for such comparisons.

6 Summary

We have introduced `pymarian`, a set of Python bindings that export Marian’s fast training and inference capabilities to Python settings, without requiring any model conversion into much slower frameworks.

These bindings enable a range of integrations with Python—the preferred language for research in NLP and MT—making available Marian’s high training and inference speeds. In particular, it enables `pymarian-eval`, an implementation of COMET and BLEURT models yielding speedups as high as $7.8\times$ (for *wmt22-cometkiwi-da*) on eight GPUs, and never less than $3.5\times$. `pymarian-eval` is also significantly faster at loading models, and up to $44x$ (for *wmt23-cometkiwi-da-xl*) on eight GPUs. These models are made available on HuggingFace and are seamlessly downloaded at runtime.

¹¹<https://github.com/OpenNMT/CTranslate2#benchmarks>

Limitations

PyMarian aims to enhance the accessibility and usability of Marian NMT and publicly available machine translation models trained with the toolkit. The primary limitation of PyMarian is that it is designed specifically for Marian-trained models, which may restrict its flexibility for users who wish to integrate models trained using other frameworks or custom architectures. Additionally, we have implemented only the most popular evaluation metrics, such as COMET and BLEURT, which may not encompass all the evaluation metrics required for specific research or application needs.

COMET-Kiwi models require users to accept a custom license and terms of use. To ensure that the license is preserved in the Marian-trained versions, we collaborated with the original authors. They now host our models exclusively on HuggingFace, where users must accept the same license before downloading. The availability of these models is subject to their decisions.

The reported benchmarks are based on specific hardware and software settings and may not fully capture the variability in real-world scenarios. Despite the optimizations, running MT evaluation metrics can be resource-intensive, requiring significant computational power. This limitation may pose challenges for users with limited access to high-performance computing resources.

Finally, as with any open-source project, the

long-term maintenance and support of PyMarian depend on the community’s contributions and engagement. Ensuring the project’s sustainability requires continuous collaboration and support from the community.

References

- Nikolay Bogoychev, Jelmer Van der Linde, and Kenneth Heafield. 2021. [TranslateLocally: Blazing-fast translation running on the local CPU](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 168–174, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Thamme Gowda, Tom Kocmi, and Marcin Junczys-Dowmunt. 2023. [Cometoid: Distilling strong reference-based machine translation metrics into Even stronger quality estimation metrics](#). In *Proceedings of the Eighth Conference on Machine Translation*, pages 751–755, Singapore. Association for Computational Linguistics.
- Kenneth Heafield, Hiroaki Hayashi, Yusuke Oda, Ioannis Konstas, Andrew Finch, Graham Neubig, Xian Li, and Alexandra Birch. 2020. [Findings of the fourth workshop on neural generation and translation](#). In *Proceedings of the Fourth Workshop on Neural Generation and Translation*, pages 1–9, Online. Association for Computational Linguistics.
- Kenneth Heafield, Biao Zhang, Graeme Nail, Jelmer Van Der Linde, and Nikolay Bogoychev. 2022. [Findings of the WMT 2022 shared task on efficient translation](#). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 100–108, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.
- Kenneth Heafield, Qianqian Zhu, and Roman Grundkiewicz. 2021. [Findings of the WMT 2021 shared task on efficient translation](#). In *Proceedings of the Sixth Conference on Machine Translation*, pages 639–651, Online. Association for Computational Linguistics.
- Felix Hieber, Michael Denkowski, Tobias Domhan, Barbara Darques Barros, Celina Dong Ye, Xing Niu, Cuong Hoang, Ke Tran, Benjamin Hsu, Maria Nadejde, Surafel Lakew, Prashant Mathur, Anna Currey, and Marcello Federico. 2022. [Sockeye 3: Fast neural machine translation with pytorch](#).
- 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. 2018a. [Marian: Fast neural machine translation in C++](#). In *Proceedings of ACL 2018, System Demonstrations*, pages 116–121, Melbourne, Australia. Association for Computational Linguistics.
- Marcin Junczys-Dowmunt, Kenneth Heafield, Hieu Hoang, Roman Grundkiewicz, and Anthony Aue. 2018b. [Marian: Cost-effective high-quality neural machine translation in C++](#). In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 129–135, Melbourne, Australia. Association for Computational Linguistics.
- Tom Kocmi, Eleftherios Avramidis, Rachel Bawden, Ondřej Bojar, Anton Dvorkovich, Christian Federmann, Mark Fishel, Markus Freitag, Thamme Gowda, Roman Grundkiewicz, Barry Haddow, Philipp Koehn, Benjamin Marie, Christof Monz, Makoto Morishita, Kenton Murray, Makoto Nagata, Toshiaki Nakazawa, Martin Popel, Maja Popović, and Mariya Shmatova. 2023. [Findings of the 2023 conference on machine translation \(WMT23\): LLMs are here but not quite there yet](#). In *Proceedings of the Eighth Conference on Machine Translation*, pages 1–42, Singapore. Association for Computational Linguistics.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. [Efficient memory management for large language model serving with pagedattention](#). In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP ’23*, page 611–626, New York, NY, USA. Association for Computing Machinery.
- Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. [fairseq: A fast, extensible toolkit for sequence modeling](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.
- 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. 2019. [Pytorch: An imperative style, high-performance deep learning library](#). In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- Matt Post. 2018. [A call for clarity in reporting BLEU scores](#). In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.
- Ricardo Rei, José G. C. de Souza, Duarte Alves, Chrysoula Zerva, Ana C Farinha, Taisiya Glushkova, Alon Lavie, Luisa Coheur, and André F. T. Martins. 2022a. [COMET-22: Unbabel-IST 2022 submission for the metrics shared task](#). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 578–585, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

- Ricardo Rei, Ana C Farinha, Chrysoula Zerva, Daan van Stigt, Craig Stewart, Pedro Ramos, Taisiya Glushkova, André F. T. Martins, and Alon Lavie. 2021. [Are references really needed? unbabel-IST 2021 submission for the metrics shared task](#). In *Proceedings of the Sixth Conference on Machine Translation*, pages 1030–1040, Online. Association for Computational Linguistics.
- Ricardo Rei, Nuno M. Guerreiro, Josã© Pombal, Daan van Stigt, Marcos Treviso, Luisa Coheur, José G. C. de Souza, and André Martins. 2023. [Scaling up CometKiwi: Unbabel-IST 2023 submission for the quality estimation shared task](#). In *Proceedings of the Eighth Conference on Machine Translation*, pages 841–848, Singapore. Association for Computational Linguistics.
- Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020a. [COMET: A neural framework for MT evaluation](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2685–2702, Online. Association for Computational Linguistics.
- Ricardo Rei, Craig Stewart, Ana C Farinha, and Alon Lavie. 2020b. [Unbabel’s participation in the WMT20 metrics shared task](#). In *Proceedings of the Fifth Conference on Machine Translation*, pages 911–920, Online. Association for Computational Linguistics.
- Ricardo Rei, Marcos Treviso, Nuno M. Guerreiro, Chrysoula Zerva, Ana C Farinha, Christine Maroti, José G. C. de Souza, Taisiya Glushkova, Duarte Alves, Luisa Coheur, Alon Lavie, and André F. T. Martins. 2022b. [CometKiwi: IST-unbabel 2022 submission for the quality estimation shared task](#). In *Proceedings of the Seventh Conference on Machine Translation (WMT)*, pages 634–645, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.
- Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. [BLEURT: Learning robust metrics for text generation](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7881–7892, Online. Association for Computational Linguistics.
- Jörg Tiedemann, Mikko Aulamo, Daria Bakshandaeva, Michele Boggia, Stig-Arne Grönroos, Tommi Nieminen, Alessandro Raganato Yves Scherrer, Raul Vazquez, and Sami Virpioja. 2023. [Democratizing neural machine translation with OPUS-MT](#). *Language Resources and Evaluation*, (58):713–755.
- Jörg Tiedemann and Santhosh Thottingal. 2020a. [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.
- Jörg Tiedemann and Santhosh Thottingal. 2020b. [OPUS-MT — Building open translation services for the World](#). In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation (EAMT)*, Lisbon, Portugal.
- Xiaolin Wang, Masao Utiyama, and Eiichiro Sumita. 2018. [Cytonmt: an efficient neural machine translation open-source toolkit implemented in c++](#). *arXiv preprint arXiv:1802.07170*.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. [Transformers: State-of-the-art natural language processing](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.