`stopes` - Modular Machine Translation Pipelines

Pierre Andrews* and **Guillaume Wenzek*** and **Kevin Heffernan*** and **Onur Çelebi***
and Anna Sun and Ammar Kamran and Yingzhe Guo
and Alexandre Mourachko and Holger Schwenk and Angela Fan

Meta AI / mortimer@fb.com

Abstract

Neural machine translation, as other natural language deep learning applications, is hungry for data. As research evolves, the data pipelines supporting that research evolve too, oftentimes re-implementing the same core components. Despite the potential of modular codebases, researchers have but little time to put code structure and reusability first. Unfortunately, this makes it very hard to publish clean, reproducible code to benefit a wider audience. In this paper, we motivate and describe `stopes` , a framework that addresses these issues while empowering scalability and versatility for research use cases. This library was a key enabler of the No Language Left Behind project, establishing new state of the art performance for a multilingual machine translation model covering 200 languages. `stopes` and the pipelines described are released under the MIT license at `https://github.com/facebookresearch/stopes`.

1 Introduction

Machine translation (MT) aims at removing language barriers in our connected society. The current trend in the MT research field is moving towards using deep machine learning models training either many bi-lingual models, translating between a single pair of languoids, or multi-lingual models that handle many languoids at once. Training data usually comes from open aligned data sources such as Barrault et al. (2020), Schwenk et al. (2021) or raw web corpora like CommonCrawl (CC). Recently, initiatives like (Bapna et al., 2022) and the No Language Left Behind project (NLLB Team et al., 2022) strive to extend the scope of supported languoids by training on large scale datasets, reaching over 18 and 25 billion sentence pairs respectively.

The end-to-end process of developing and iterating on a neural machine translation model involves a lot of large scale steps. Getting large amounts of data prepared for translation training usually starts with raw monolingual data composed of unaligned sentences for each languoid of interest. This web data is usually processed, cleaned, and finally "mined" to be aligned in pairs of translated sentences (see 5.2). It is then tokenized and transformed into a format that can be used for training. Once trained, the machine translation model is evaluated using benchmark datasets, on which the exact same pre-processing has to be applied. Large translation models are often later distilled to produce smaller models suitable for practical production usecases (see 5.3).

In research use cases, the main focus is on getting results fast. We have observed that the path of predilection is to build ad-hoc solutions to solve the problem directly at hand, often adapting older scripts or copying snippets of code that colleagues have found to work. This enables quick iteration on research ideas but it causes a lot of problems on the long term:

1. Scaling and data processing throughput is often an after-though.
2. Ad-hoc scripts are built with the idiosyncrasies of each users and experiment, making it hard to share research pipelines or adapt to different hardware setups.
3. Open-sourcing research and making it reproducible by third parties becomes a task of itself, ensuring scripts will run properly in different environment and without failed experimental code/setup (Pineau et al., 2021; Ulmer et al., 2022).
4. When working with disjoint scripts, researchers spend a lot of time "baby sitting" execution, making sure one script runs properly and waiting for it to finish before moving to the next step in their pipeline.

*Lead Library Maintainers.

Proceedings of the The 2022 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 258 - 265

We present a new framework, `stopes` , that was developed to solve some of the problems discussed in the scope of the No Language Left Behind (NLLB) (NLLB Team et al., 2022) machine translation project to process billions of sentences in over 200 languages. The goal of the framework is to ensure a good separation between the hardware setup and the core logic of the data processing by proposing a clean API for sharing commonly used processing steps, while enforcing consistent and shareable configurations of experiments. `stopes` can scale to a research project like NLLB, but is built to be versatile and can be applied to other research use cases, `https://facebookresearch.github.io/stopes/docs/quickstart` provides an example of running this on a smaller dataset. In section 3, we introduce the design of this python library, with concrete examples in section 4. Section 5 discusses applications within the NLLB where `stopes` is used.

2 Related Work

While large scale data processing architectures already exist, they are often optimized for production use cases. Spark (Spark), ray (Ray) or beam (Beam) are a few leading examples. These frameworks have a steep learning curve and do not always map easily to research clusters' setup or researchers' work habits. They can also prove very challenging to use with nascent research ideas and tools, whose codebases are not yet stable or production-ready. Bitextor (Bitextor) provides a bitext mining pipeline built in python, but it lacks modularity and requires learning the complex APIs of snakemake.

With `stopes` , we are aiming for a "minimal API surface" without sacrificing features, providing a clean yet versatile API that can be used as if writing standard python scripts (see Section refsec:example). This simple API has its drawbacks, but it makes it easier to pick up for researchers than complex graph planning systems like Luigi (Luigi) and AirFlow (AirFlow), These industry standards are better suited for production pipelines that do not change often and are maintainted by production teams. Spacy (Spacy) provides research oriented NLP pipelines, but is less flexible than `stopes` as our framework is more geared towards describing sometimes pipelines in pure python.

3 Framework

The general architecture of `stopes` is geared towards pipelines that can be run as separate, sometimes interdependent, jobs on a cluster or in multiprocessing. The idea being that a pipeline can be divided in a set of separate steps that can be expressed as processing units. Jobs can be sent to a job scheduler, like SLURM (Slurm), which is widespread on academic compute clusters or FBLearner (Dunn, 2016), which Meta uses for distributed machine learning pipelines; or run locally on a single computer depending on the data scale.

The idea behind the `stopes` framework is to make it easy to build reproducible pipelines. This is done though *modules*, a module is just a python class with a `run` function that executes something. A module can then be scheduled with the `stopes` ' *launcher*, this will decide where the code gets executed (locally or on a cluster) and then wait for the results to be ready.

3.1 Concepts

module: Encapsulate a reusable single step of a neural network pipeline and its requirements. The step is assumed to be able to execute on its own given some inputs and eventually generates an output. Modules will most often be executed as an isolated job, so should not depend on anything other than its own configuration (e.g. no global variables or odd i/o dependencies). This ensures that each module can be ran separately, or in parallel if possible. A module's configuration serves the purpose of defining a clear API of the step.

pipeline: A python function which connects `stopes` modules together for some end-to-end purpose. Pipelines may contain non-module logic to help with intermediate functionality, and are primarily structured like functions as opposed to `stopes` modules which resemble python callables. In some cases, pipelines may also call other pipelines in intermediate steps.

launcher: The orchestrator of your pipeline. The power of `stopes` comes from the *launcher* that will manage the execution of the modules, find the correct machines with matching requirements (if executing on a cluster), and deal with memoization (see below). The launcher abstracts the execution/scheduling of modules as it looks like any asyncio function and can be called like a python function and utilized in conjunction with regular python code.

3.2 Configuration

When running experiments in machine translation, we often change how the data is processed or what data we ingest. For instance, we might want to change the vocabulary size, which would require retraining a tokenization model (e.g. sentence-piece[1] or BPE (Sennrich et al., 2016)). To keep track of experiments and ensure reproducibility, all parameters that can influence the results need to be stored in configuration files that can easily be shared with other researchers.

`stopes` makes it easy to keep track of configurations as it leverages the hydra configuration system (Yadan, 2019) as inputs for modules and pipelines. This guarantees proper tracking of configurations through the execution of a pipeline, but also brings extra technical benefits to the end user:

1. New configurations can be composed from existing configuration files, allowing for better organization of all steps within a pipeline.
2. Any part of a configuration can be overridden at runtime and across multiple runs. This makes it easy, for example, to change what cluster the code is running on, what model architecture is used for training, or what tokenization approach is used.

3.3 Caching/Memoization

As we can see in Section 5, machine translation research pipelines are complex and involve a lot of steps. When repeating these steps over many languoids, some of the jobs executing the pipeline are bound to fail. Failure is common when executing large pipelines over long periods of time, jobs might timeout in the cluster queue, disk might fail because of IO pressure and machines might go down for maintenance.

It is therefore very important to be able to re-run a pipeline over and over and not have to start from the scratch. To avoid this, `stopes` memoizes the output of each module runs based on its input configuration. If the module is re-run with the same configuration, `stopes` will recover the results from disk instead of re-running. This can be seen as a cache of the results, indexed on the input configuration of each module. The exact cache invalidation logic can be manually tuned by the user to accommodate more complex situations.

This is also very practical when iterating on configuration driven experiments as `stopes` will figure out automatically what steps of the pipeline needs to be re-run when the configuration changes, keeping track of identical steps in the pipeline that were not affected by the experimental configuration change and re-using cached results.

4 Example Code

Figure 1 shows a sample usage of the `stopes` library[2] to build a FAISS index[3]. FAISS (Johnson et al., 2019) is a tool that can be used to build large scale indexes and perform nearest neighbor searches on them; FAISS has become a keystone to machine translation research as it allows for efficient alignment of multilingual text when using language-agnostics embeddings like Feng et al. (2020) or Heffernan et al. (2022) (e.g. Khandelwal et al. (2021) or Section 5.2). It takes tensors as input, but first the index has to be trained, usually on a sample of the data we want to store in the index.

Line 5: We initialize the `launcher` to be able to schedule modules for execution. The launcher is managed by a configuration, so we can easily change where the code is executed (SLURM cluster, aws, locally) and other constraints of the execution. Every call to 'launcher.schedule' will be managed by the designated 'launcher', sent to the cluster once, or in multiple jobs if necessary, or just retrieved from the cache if the config permits.

Line 7: We initialize an encoding module, which takes in raw text and embeds it. `stopes` provides code to embed text with LASER2 and LASER3 as well as with HuggingFace `sentence-transformers` (Reimers and Gurevych, 2019). To keep the code short, we only show the pipeline glue and not each module implementation.

Line 12: We create a sample from the embedded text files. Here, `update(config.sample, input_embeddings=embedded)` takes the module configuration from the Hydra configuration and inserts the references to the output files from the previous pipeline step. This pattern can also be seen in the other steps of this pipeline where each step is connected to the previous through intermediate output results.

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

[2] Modules referred to in the sample code can be found in the `stopes` open source repository.

[3] We have omitted the imports from the sample to keep it short.

```python
# ... imports omitted
async def pipeline(config):
    # setup a launcher to connect jobs together
    launcher = hydra.utils.instantiate(config.launcher)
    # encode all shards
    embedded = await
        launcher.schedule(PreprocessEncodeModule(config=config.embed_text))
    # extract a sample of the embeddings
    train_sample = await launcher.schedule(
        SampleEmbeddingModule(config=update(config.sample, input_embeddings=embedded))
    )
    # train the faiss index on the sample
    trained_index = await launcher.schedule(
        TrainFAISSIndexModule(
            config=update(config.train_index, input_embeddings=train_sample)
        )
    )
    # fill the index with content
    populated_index = await launcher.schedule(
        PopulateFAISSIndexModule(
            config=update(
                config.populate_index,
                index=trained_index,
                input_embeddings=embedded,
            )
        )
    )
    print(f"Indexes are populated in: {populated_index}")


# setup main with Hydra
@hydra.main(config_path="conf", config_name="config")
def main(config: DictConfig) -> None:
    asyncio.run(pipeline(config))
```

Figure 1: Sample pipeline to build a FAISS Index with `stopes`

Lines 17 and 22: These lines use very similar logic to call different modules. As noted above, we can see the use of the configurations passed by Hydra extended with the results from the previous steps.

From this, we see that `stopes` pipeline code reads as normal python code where functions are called and pass results to each other. The core of `stopes` hides the complexity of memoization and cluster scheduling inside the simple API call to `launcher.schedule`. This makes the pipeline easy to understand and allows researcher to focus on building data processing and stay close to their research goals instead of getting bogged down in boilerplate APIs or in optimization/scaling issues.

5 Applications

The `stopes` library was used to build the major data processing pipelines that are used to build the NLLB large multilingual translation models as well as its distilled version (NLLB Team et al., 2022). These pipelines were battle tested on petabytes of data and are open-sourced at https://github.com/facebookresearch/stopes. In this section we discuss some of the pipelines and show how they can reuse the same modules. Source code can be found in the above github repository.

5.1 Language Identification

The production of large amount of monolingual data starts with a strong language identification (LID) model (see NLLB Team et al., 2022). The pipeline for training an LID model is a recurring archetype used commonly in neural network training pipelines for machine translation. LASER3 distillation (Heffernan et al., 2022), training NMT models for evaluation, etc., all use a similar pipeline.

The pipeline is illustrated in Figure 2 and uses the following steps:

SPM Training: Eventually, we will use a sentence-piece model (SPM) to tokenize input data for neural network training. To be able to do this tokenization, the SPM itself must be first trained

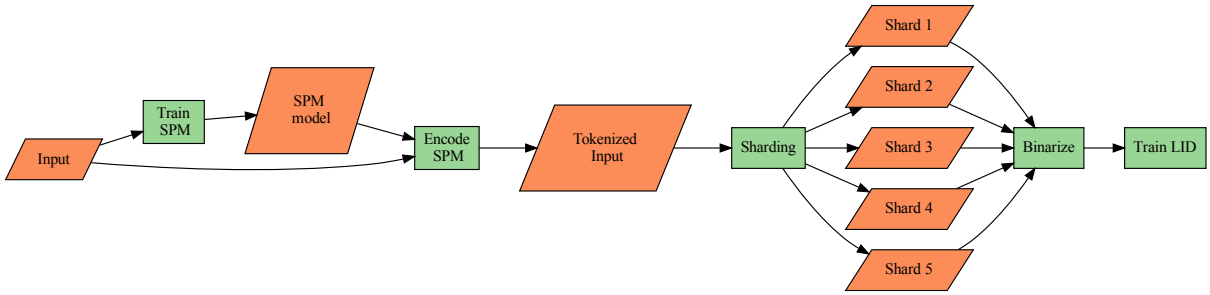

Figure 2: LID Model Training Pipeline

on a sample of data.

SPM Encoding: Once we have a trained SPM, we can apply it over all the sentences in the input data to tokenize the raw text to prepare it for training.

Sharding: This step is used to split the data into manageable shards that help distribute the pipeline work over multiple jobs, and also at the training phase to be able to fit the training data in the memory available on each training machine.

Binarization: The SPM tokenization process creates tokenized text, but the model training loop requires numerical tensors to do the neural network training. The binarization process takes each token and the SPM vocabulary to create binary tensors from the tokenized text.

Model Training: We use `fairseq` (Ott et al., 2019) and fastText (Joulin et al., 2017) to train LID models and other NMT models.

5.2 Bitext Mining

The bitext mining pipeline follows the idea introduced by Schwenk et al. (2021). The pipeline can mine pairs of sentences between two languoids given monolingual data and evaluate the mining quality. It follows the following major steps:

Monolingual Data: The base data comes from a mix of existing "clean" data (Barrault et al., 2020) and noisy web data. Most of this data is not aligned in language pairs, and often not tagged with a particular languoid. The monolingual pipeline runs a language identification (LID) model, splits text into sentences, and then cleans the text. The LID model itself is trained as discussed in Section 5.1

FAISS Indexing: FAISS (Johnson et al., 2019) is a tool to build large indexes for similarity searches. Section 4 shows a sample pipeline to build such an index. In bi-text mining, the FAISS index serves as the core tool to find similar sentences between two languoids. This works by filling the index with sentences embedded with LASER3 (Heffernan et al., 2022), which encodes sentences from different languages into the same space, so they can be clustered by FAISS. The mining pipeline then builds a separate index for each languoid, embedding all the sentences identified in the monolingual data, sampling them to train a FAISS index (i.e. to learn the clustering), and then populating the index with all the embedded sentences for that langoid.

Mining for Aligned Sentences: To align sentences between two languoids, we go over all embedded sentences from one languoid and use the cosine distances of the k-nearest neighbors in the other languoid index compute above and output alignments using a margin-based scoring measure (Artetxe and Schwenk, 2019). Once we've built indexes and embeddings for a few languages, we can run this step in parallel quite easily. `stopes` makes this trivial as it will pickup the embeddings and indexes from its cache and jump straight to the last step of the mining pipeline, without the user having to figure out what has already been pre-computed.

Evaluating Translation: There is no direct evaluation procedure to gauge mining quality. Therefore, the best way to evaluate the mining performance is to use the aligned bitext it produces to train a neural machine translation model. We focus on training bi-lingual translation models as they are faster to train and evaluate. We can then track the change in BLEU score (Papineni et al., 2002) for a given model and languoids pair to evaluate the specialized encoders and mining parameters.

Figure 3 illustrates the high level process of mining for two languoids. The figure shows the process for two languoids, but when mining for training a large language model as the one discussed in NLLB Team et al. (2022), we ran this `stopes` pipeline over 450 pairs, aligning over a billion sentences.

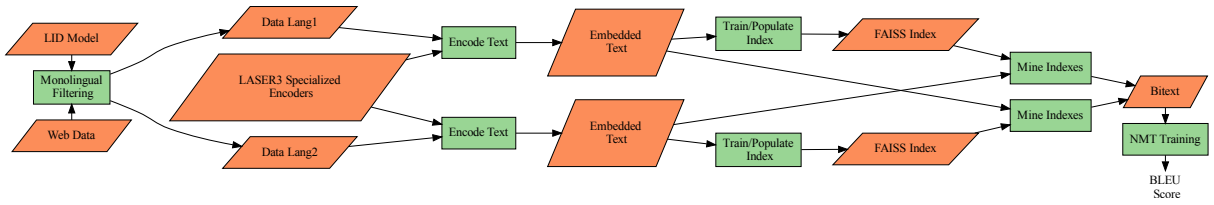

Figure 3: Mining Pipeline

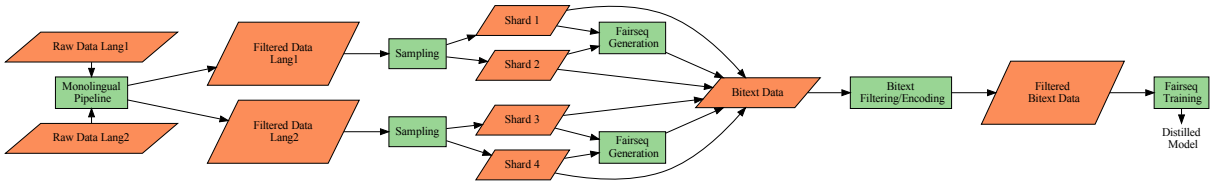

Figure 4: Distillation Pipeline

This is where the strong configuration system introduced by `stopes` comes handy as we need to manage different configurations for over two thousand pairs of languoids. Being able to write the pipeline once and scale it to many languoids through simple configuration composition and horizontal scaling on a SLURM cluster without having to rewrite core logic greatly accelerates the speed at which research is conducted. The pipeline that was used by the NLLB project is available on the `stopes` repository and can be run by anyone a "small" scale to mine data with our approach[4].

5.3 Large Model Distillation

The distillation pipeline is based on the sequence-level knowledge distillation proposed by Kim and Rush (2016), using a large pre-trained teacher model to help train a smaller student model with comparable or better performance, which is practical for inference efficiency. An overview of the steps are visualized in Figure 4 and described below:

Monolingual Data: Our monolingual source data comes from Wikipedia corpus dumps[5]. The monolingual pipeline is the same as the one described in Section 5.2.

Sampling: We sample with replacement from the monolingual dataset to ensure that we have enough target sentences for each languoid, given a fixed-size monolingual source dataset.

Generation: We use the `FairseqGenerate` module to generate translations of each shard of monolingual data by running beam search using the teacher model.

Bitext Filtering: We filter the teacher-generated bitext data to make sure the training data is high quality. We use LID and sentence length filtering to ensure that the generated data matches the target languoid and that the sentence lengths are similar.

SPM Encoding: We use a pre-trained SPM to tokenize the raw text.

Binarization: We binarize the bitext data into the format required for training in `fairseq` as described in Section 5.2.

Training: Using the binarized bitext data, we use the module `TrainFairseqModule` to train a multilingual distilled model, the final product of our pipeline.

6 Conclusion

The `stopes` framework provides a clean API to describe research pipelines for machine translation. We have shown that this is useful for developing large scale machine translation datasets and models for the No Language Left Behind project (NLLB Team et al., 2022). We believe that this framework and its reference implementations of common steps in NLP pipelines is versatile and can be used to help researchers in the field. The `stopes` framework documentation and sources can be found under the MIT license at `https://facebookresearch.github.io/stopes/` and has been tested to not require a complex cluster setup. We therefore hope that it will help other researchers focus on their research goals, and avoid time-consuming technical details not unique to their specific task.

[4] See the quickstart at `https://facebookresearch.github.io/stopes/docs/quickstart`

[5] `https://dumps.wikimedia.org/other/cirrussearch/current/`

7 Screencast Video

The demo screencast can be found at https://fb.sharepoint.com/:f:/s/PublicContent/EsuaUW__krBJo57yDgbbbysBw5yN5txcRsjw4eYlYRFIFQ?e=tAlkVQ.

References

Airflow. https://github.com/apache/airflow. Accessed: 2022-07-19.

Beam. https://beam.apache.org/. Accessed: 2022-07-19.

Bitextor. https://github.com/bitextor/bitextor. Accessed: 2022-07-19.

Common crawl. https://commoncrawl.org/. Accessed: 2022-07-19.

Luigi. https://github.com/spotify/luigi. Accessed: 2022-07-19.

Ray. https://www.ray.io/. Accessed: 2022-07-19.

Slurm. https://slurm.schedmd.com/. Accessed: 2022-07-19.

Spacy. https://spacy.io/. Accessed: 2022-07-19.

Spark. https://spark.apache.org/. Accessed: 2022-07-19.

Mikel Artetxe and Holger Schwenk. 2019. Margin-based parallel corpus mining with multilingual sentence embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3197–3203, Florence, Italy. Association for Computational Linguistics.

Ankur Bapna, Isaac Caswell, Julia Kreutzer, Orhan Firat, Daan van Esch, Aditya Siddhant, Mengmeng Niu, Pallavi Baljekar, Xavier Garcia, Wolfgang Macherey, Theresa Breiner, Vera Axelrod, Jason Riesa, Yuan Cao, Mia Xu Chen, Klaus Macherey, Maxim Krikun, Pidong Wang, Alexander Gutkin, Apurva Shah, Yanping Huang, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2022. Building machine translation systems for the next thousand languages.

Loïc Barrault, Magdalena Biesialska, Ondřej Bojar, Marta R. Costa-jussà, Christian Federmann, Yvette Graham, Roman Grundkiewicz, Barry Haddow, Matthias Huck, Eric Joanis, Tom Kocmi, Philipp Koehn, Chi-kiu Lo, Nikola Ljubešić, Christof Monz, Makoto Morishita, Masaaki Nagata, Toshiaki Nakazawa, Santanu Pal, Matt Post, and Marcos Zampieri. 2020. Findings of the 2020 conference on machine translation (WMT20). In *Proceedings of the Fifth Conference on Machine Translation*, pages 1–55, Online. Association for Computational Linguistics.

Jeffrey Dunn. 2016. Introducing FBLearner flow: Facebook's ai backbone. https://engineering.fb.com/2016/05/09/core-data/introducing-fblearner-flow-facebook-s-ai-backbone/. Accessed: 2022-07-19.

Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. 2020. Language-agnostic BERT sentence embedding. *CoRR*, abs/2007.01852.

Kevin Heffernan, Onur Çelebi, and Holger Schwenk. 2022. Bitext mining using distilled sentence representations for low-resource languages.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics.

Urvashi Khandelwal, Angela Fan, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2021. Nearest neighbor machine translation. In *International Conference on Learning Representations*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. 2022. No language left behind: Scaling human-centered machine translation.

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 NAACL-HLT 2019: Demonstrations*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. pages 311–318.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. 2021. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *J. Mach. Learn. Res.*, 22(1).

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Holger Schwenk, Guillaume Wenzek, Sergey Edunov, Edouard Grave, Armand Joulin, and Angela Fan. 2021. CCMatrix: Mining billions of high-quality parallel sentences on the web. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6490–6500, Online. Association for Computational Linguistics.

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.

Dennis Ulmer, Elisa Bassignana, Max Müller-Eberstein, Daniel Varab, Mike Zhang, Christian Hardmeier, and Barbara Plank. 2022. Experimental standards for deep learning research: A natural language processing perspective.

Omry Yadan. 2019. Hydra - a framework for elegantly configuring complex applications. Github.