Lego-MT: Learning Detachable Models for Massively Multilingual Machine Translation

Fei Yuan[1], **Yinquan Lu**[1], **Wenhao Zhu**[2], **Lingpeng Kong**[3], **Lei Li**[4], **Yu Qiao**[1], **Jingjing Xu**[1]
[1] Shanghai Artificial Intelligence Laboratory
[2] National Key Laboratory for Novel Software Technology, Nanjing University, China
[3] The University of Hong Kong, [4] University of California, Santa Barbara
{yuanfei, luyinquan, qiaoyu}@pjlab.org.cn, zhuwh@smail.nju.edu.cn
lpk@cs.hku.hk, leili@cs.ucsb.edu, jingjingxupku.02@gmail.com

Abstract

Multilingual neural machine translation (MNMT) aims to build a unified model for many language directions. Existing monolithic models for MNMT encounter two challenges: parameter interference among languages and inefficient inference for large models. In this paper, we revisit the classic multi-way structures and develop a detachable model by assigning each language (or group of languages) to an individual branch that supports plug-and-play training and inference. To address the needs of learning representations for all languages in a unified space, we propose a novel efficient training recipe, upon which we build an effective detachable model, Lego-MT. For a fair comparison, we collect data from OPUS and build a translation benchmark covering 433 languages and 1.3B parallel data. Experiments show that Lego-MT with 1.2B parameters brings an average gain of 3.2 spBLEU. It even outperforms M2M-100 with 12B parameters. The proposed training recipe brings a 28.2× speedup over the conventional multi-way training method.[1]

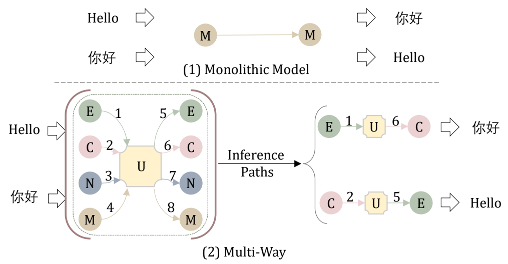

Figure 1: Multi-way architecture. *(1) Monolithic Model* is the fully-shared model for all translation directions; *(2) Lego-MT* is a multi-way structure that includes both multilingual (denoted as M) and language-specific encoders and decoders for English (denoted as E), Chinese (denoted as C) and Nepali (denotes as N). The architecture is detachable at inference time where only a specific encoder and decoder are needed. U (Unified space) represents hidden representations generated by encoders.

1 Introduction

Multilingual neural machine translation (MNMT) translates languages by mapping a source sentence to a unified representation space and decoding a target sentence from this space (Johnson et al., 2017; Gu et al., 2018; Neubig and Hu, 2018; Aharoni et al., 2019; Zhang et al., 2020). Traditional MNMT models use a shared network to align representations in different languages. Recently, scaling up the size of MNMT models brings significant quantitative improvements and new qualitative capabilities (M2M-100, Fan et al. 2021; NLLB-200, Costa-jussà et al. 2022; *inter alia*). Beyond MNMT, recent large-scale language models (e.g., ChatGPT) also show promising results on zero-shot (or few-shot) translation, especially for language-to-English translation. Despite great potential, there is still a large gap between LLMs and existing MNMT models on massive translation directions.

Simply using a shared model for massive MNMT brings new effectiveness and efficiency issues. First, memorizing multilingual knowledge within finite parameters causes parameter interference (Ha et al., 2016a), especially between high-resource and low-resource languages (Li and Gong, 2021), which leads to significant performance degradation. Second, the centralization feature requires all parameters to be included in the computation graph during the inference stage, resulting in heavy computational overhead (Song et al., 2021). Common fixes of these issues include adapter-based approaches (Zhu et al., 2021), which handle parameter interference via fine-tuning new parameters to fit bilingual translation, and mixture-of-expert (MoE), which supports dynamic activation. These methods either fail to adapt to massive translation directions or require all parameters to be loaded into memory, thus remaining unsatisfactory consid-

[1] https://github.com/CONE-MT/Lego-MT.

ering the efficiency of training and inference.

To find out the best recipe for massive multilingual translation, we revisit the classic multi-way (or multi-branch) architecture (Dong et al., 2015; Firat et al., 2016), whose philosophy is to allocate an individual encoder and decoder for each language (or group of languages), as shown in Figure 1. The immediate benefit of this structure is: 1) The utilization of individual modules for specific languages mitigates parameter interference; 2) Each branch can be independently loaded during inference, significantly reducing computational costs and decreasing inference latency.

Despite appealing, there remain two big challenges when training multi-way structures: *representation alignment* between different languages due to the lack of shared parameters; and *low GPU efficiency* during training because unused parameters occupy GPU memory but do not have any computations. Furthermore, the feature of random language mixture in a batch makes it infeasible to use an online-loading method (i.e., loading during usage) to accelerate training since it will cause impractical IO communication costs during batch switching (between CPU and GPU).

To address these challenges, we propose a novel training recipe, which results in our new detachable model, Lego-MT. We classify the training data into different language-centric groups such that we only need to load specific branches into GPU memory, eliminating the need to load different modules constantly. The language-centric group is trained in sequential order. Second, during each language-centric training, we introduce a multilingual branch and propose a new triple-flow method to help a model learn to map to and translate from a unified space. Specifically, a unified space is a type of representation space rather than a module. It creates a common representation of language that can be used across multiple language tasks.

To evaluate our training recipe for massive MNMT, we construct a many-to-many translation dataset[2] covering 7 language-centric groups, 433 languages, based on the open-source website OPUS[3] (Tiedemann, 2012).

Lego-MT-1.2B yields average gains of 3.2 spBLEU, and even outperforms M2M-100-12B which has 10× inference parameters. Furthermore, the proposed training recipe brings a 28.2× speedup compared with the conventional multi-way training method. We also conduct comprehensive experiments on branch combinations, thanks to the detachable nature of the model. We find that low-resource languages prefer multilingual branches and high-resource languages prefer language-specific branches. In addition, we also observe that the unseen combination of a high-resource language encoder and a high-resource language decoder can achieve better performance, showing that Lego-MT can align different branches into a unified space effectively. The main contributions can be summarized as follows:

- We build an effective detachable model Lego-MT for multilingual machine translation.
- Experiments demonstrate that Lego-MT brings an average gain of 3.2 spBLEU. This training recipe results in a 28.2× training speedup compared with the naive multi-branch architecture.
- We construct a massive multilingual translation dataset covering 433 languages, which greatly extends the scale of languages.

2 Related Work

In this part, we review recent related multilingual machine translation models. We classify them into three categories: fully / group-shared (Dabre et al., 2020), and Mixture-of-expert (MoE).

The fully-shared model is the most prevalent model in Multilingual Neural Machine Translation (MNMT). This model employs a single architecture to translate in all directions (Ha et al., 2016b; Johnson et al., 2017; Bapna et al., 2019; Lin et al., 2020; Liu et al., 2020; Pan et al., 2021; Sun et al., 2021) and has demonstrated efficacy in aiding low-resource directions. However, fully-shared models are often subject to capacity bottlenecks and trade-offs between translation quality and the number of languages (Aharoni et al., 2019; Zhang et al., 2020; Ha et al., 2016a). Group-shared models incorporate individual parameters for each group and represent a popular solution for sharing language-specific encoders or decoders (Lee et al., 2017; Zoph and Knight, 2016). Lee et al. (2017); Sachan and Neubig (2018); Ji et al. (2020); Lyu et al. (2020); Sachan and Neubig (2018) proposed an MNMT model only for shared language-specific modules. LaSS (Lin et al., 2021) learns language-specific sub-networks for each language direction for multilingual translation. Adpater methods (Bapna and Firat, 2019; Zhu et al., 2021)

[2] The dataset is released on https://github.com/CONE-MT/Lego-MT.git.

[3] https://opus.nlpl.eu.

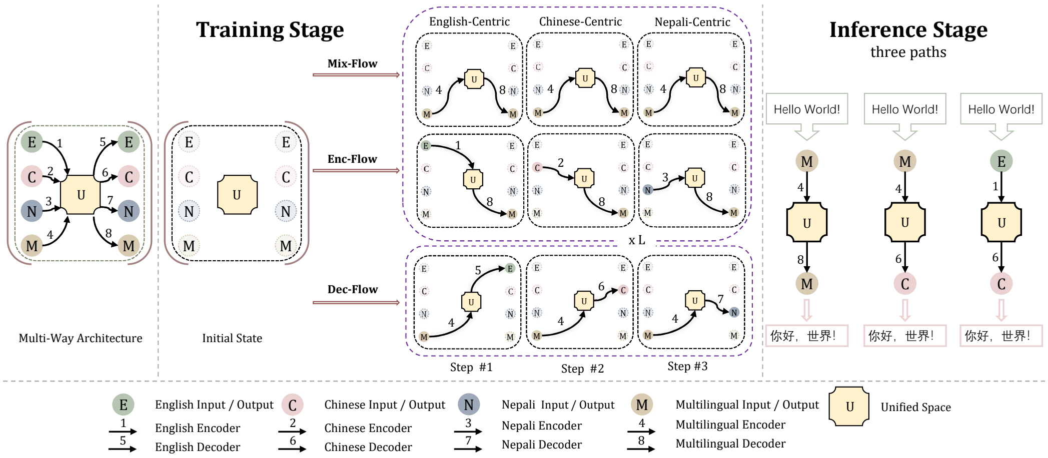

Figure 2: The overview of Lego-MT and training recipe. During training, we introduce an efficient training method by classifying multilingual data into language-centric groups. The language-centric groups are trained in a sequential way. During each training phase, only language-specific parameters are loaded into GPU memory. The training maintains three flows: *Enc-Flow* (language-specific encoder + multilingual decoder) for training specific encoder, *Dec-Flow* (Multilingual encoder + language-specific decoder) to train language-specific decoder, and *Mix-Flow* (multilingual encoder + multilingual decoder) to avoid the overfitting of multilingual encoder and decoder to each language-centric training data. U means the unified space, hidden representations generated by encoders.

add additional side networks to each language direction in addition to the main multilingual Transformer encoder-decoder. While these studies can alleviate the capacity bottleneck to some extent, challenges remain when handling larger-scale languages.

Mixture-of-Expert (MoE) has recently emerged as a prominent research direction (Jacobs et al., 1991; Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2021; Du et al., 2022; Fan et al., 2021; Costa-jussà et al., 2022), which are sparsely activated, with each inference only activating a subset of parameters. Researchers have applied MoE to massively multilingual translation and introduced various regularization strategies to enhance performance (Dai et al., 2022; Costa-jussà et al., 2022). Despite promising results, MoE's objective differs from ours, as it still requires the entire structure to be stored in GPU memory during inference.

The encoder-decoder structure has demonstrated considerable flexibility through the utilization of the Lego-NN (Dalmia et al., 2022). The Lego-NN can be applied to various tasks with decoder modules being detachable, in contrast, the Lego-MT model design allows for the performance of massively MNMT with **all modules** being detachable.

3 Lego-MT

3.1 Overview

This paper aims to build a detachable multi-branch model with a language (or group)-specific encoder and a language (or group)-specific decoder. As shown in Figure 2, the detachable structure provides an effective mechanism to only load a part of modules during training and inference.

During training, we introduce a new training method by classifying multilingual data into language-centric groups. During each training phase, only language-centric data and related branches are loaded. All language-centric groups are trained in a sequential way. We empirically found that the orders contribute little to the final performance and we fix the training order for simplification in the next parts.

During each language-centric training phase, we introduce a multi-lingual branch to help language-specific branches learn to encode to a unified space and decode from a unified space. **Unified Space** is a concept that aims to map all languages into a unified representation space without any parameters. This concept is used in natural language processing and machine learning to create a common representation of language (Lyu et al., 2020; Fan et al., 2021) that can be used across different languages.

The training maintains triple-flow: *Enc-Flow* (language-specific encoder + multilingual decoder) for training specific encoder, *Dec-Flow* (multilin-

Algorithm 1: Triple-flow training.

Input: Epoch number L. Training data for Mix-Flow, Enc-Flow and Dec-Flow: $\mathcal{D}_{\text{multi}} = \{\mathcal{D}_{s_1\to t_1}, \mathcal{D}_{s_i\to t_j}, ..., \mathcal{D}_{s_N\to t_N}\}$ and $\mathcal{D}_{\text{lg}\to\cdot} = \{\mathcal{D}_{\text{lg}\to t_1}, \mathcal{D}_{\text{lg}\to t_j}, ..., \mathcal{D}_{\text{lg}\to t_N}\}$ and $\mathcal{D}_{\cdot\to\text{lg}} = \{\mathcal{D}_{s_1\to\text{lg}}, \mathcal{D}_{s_i\to\text{lg}}, ..., \mathcal{D}_{s_N\to\text{lg}}\}$, respectively. The parameters used for Mix-Flow and Enc-Flow are initialized as $\theta_m = \theta_0$ and $\theta_e = \theta_0$. Note, the parameters used for Dec-Flow are initialized as $\theta_d = \theta_m$ after training of Mix-Flow and Enc-Flow.

for *epoch* $l = 1$ *to* L **do**
  Shuffle $\mathcal{D}_{\text{lg}\to\cdot}$ to obtain a new training sequence.
  for *each batch* $\mathcal{D}_e \in \mathcal{D}_{\text{lg}\to\cdot}$ **do**
    Evaluate the objective by Equation 2 on $\mathcal{D}_e$: $l_e = \sum_{\mathbf{x},\mathbf{y}\sim\mathcal{D}_b} -\log P_{\theta_e}(\mathbf{y}|\mathbf{x})$
    Get a minibatch of multilingual data $\mathcal{D}_m \in \mathcal{D}_{\text{multi}}$
    Evaluate the objective by Equation 1 on $\mathcal{D}_m$: $l_m = \sum_{\mathbf{x},\mathbf{y}\sim\mathcal{D}_m} -\log P_{\theta_m}(\mathbf{y}|\mathbf{x})$
    Update θ_m and θ_e by: $\theta_m \leftarrow \theta_m - \eta \bigtriangledown_{\theta_m} (l_m + l_e)$ and $\theta_e \leftarrow \theta_e - \eta \bigtriangledown_{\theta_e} l_e$
  end
end
for *epoch* $l = 1$ *to* L **do**
  Shuffle $\mathcal{D}_{\cdot\to\text{lg}}$ to obtain a new training sequence.
  for *each batch* $\mathcal{D}_d \in \mathcal{D}_{\cdot\to\text{lg}}$ **do**
    Calculate $\mathcal{D}_d$ by Equation 3 : $l_d = \sum_{\mathbf{x},\mathbf{y}\sim\mathcal{D}_d} -\log P_{\theta_d}(\mathbf{y}|\mathbf{x})$
    Update θ_d: $\theta_d \leftarrow \theta_d - \eta \bigtriangledown_{\theta_d} l_d$
  end
end

gual encoder + language-specific decoder) to train language-specific decoder, and *Mix-Flow* (multilingual encoder + multilingual decoder) to avoid the overfitting of multilingual encoder and decoder to each language-centric training data. Surprisingly, we find that Dec-flow cannot be trained together with Mix/Enc-flow, resulting in catastrophic forgetting in the multilingual encoder (detailed discussion in Section 5). Therefore, the basic training processes can be briefly divided into two stages: the Mix/Enc-Flow phase and the Dec-Flow phase.

During inference, there are three alternative flows in Lego-MT for language-centric translation to be translated ("Inference Stage" in Figure 2). As shown in Figure 2, users can decide to choose which path for inference.

3.2 Triple-Flow Training

Given a multilingual dataset with N languages, $\mathcal{D}_{\text{multi}} = \{\mathcal{D}_{s_1\to t_1}, \mathcal{D}_{s_i\to t_j}, ..., \mathcal{D}_{s_N\to t_N}\}$, where each $\mathcal{D}_{s_i\to t_j}$ contains a parallel data from the source language S_i to the target language T_j, s_i refers to the i-th ($i \in N$) language being translated from, t_j represents the j-th ($j \in N$) language being translated into, respectively. Specifically, one-to-many multilingual data for a specific language (lg) can be expressed as $\mathcal{D}_{\text{lg}\to\cdot} = \{\mathcal{D}_{\text{lg}\to t_1}, \mathcal{D}_{\text{lg}\to t_j}, ..., \mathcal{D}_{\text{lg}\to t_N}\}$. Similarly, the many-to-one multilingual data for a specific language (lg) can be denoted as $\mathcal{D}_{\cdot\to\text{lg}} = \{\mathcal{D}_{s_1\to\text{lg}}, \mathcal{D}_{s_i\to\text{lg}}, ..., \mathcal{D}_{s_N\to\text{lg}}\}$. All input sequence is preceded by a special tag (called the language tag) to indicate the source language and target languages. During each training phase, we have triple-flows playing for different rules, Mix-Flow, Dec-Flow, and Enc-Flow.

3.2.1 Mix-Flow

Mix-Flow is built upon a multilingual encoder branch and a multilingual decoder branch. It is trained on multilingual to multilingual data. This flow learns a mapping function f from a sentence in any language to another language. All language data is mixed together. The input source sequence is preceded by a special tag (called the language tag) to indicate the source languages. Following traditional methods, we also add a target language tag in the decoder part. The training loss for a Mix-Flow is:

$$\mathcal{L}_m = -\sum_{\mathbf{x},\mathbf{y}\sim\mathcal{D}_{\text{multi}}} \log P_{\theta_m}(\mathbf{y}|\mathbf{x}) \quad (1)$$

where $\mathbf{x},\mathbf{y}$ is a pair sampled from multilingual training data. It is used to avoid over-fitting language-specific data in Enc-Flow and Dec-Flow.

3.2.2 Enc-Flow

Enc-Flow includes a language-specific encoder and a multilingual decoder. It is trained with one-to-many multilingual data. The structure of such a design is natural for language-specific encoder training: the encoder input data comes from the same source language lg, and the decoder is multi-lingual data. The language tag is also added to the encoder and decoder parts. The training loss for language-specific Enc-Flow is:

$$\mathcal{L}_e = -\sum_{\mathbf{x},\mathbf{y}\sim\mathcal{D}_{\mathrm{lg}\rightarrow\cdot}} \log P_{\theta_e}(\mathbf{y}|\mathbf{x}) \tag{2}$$

where $\mathbf{x}, \mathbf{y}$ is a pair sampled from one-to-many training data.

3.2.3 Dec-Flow

Dec-Flow includes a multilingual encoder and a language-specific decoder. It is trained with many-to-one translation. We separate the training of Dec-Flow from the training of Enc-Flow and Mix-Flow. The parameters used for training Dec-Flow are initialized with the latest model trained by Mix-Flow and Enc-Flow. The language tag is also added to the encoder and decoder parts. Given a many-to-one dataset $\mathcal{D}_{\cdot\rightarrow\mathrm{lg}}$, the training loss is:

$$\mathcal{L}_d = -\sum_{\mathbf{x},\mathbf{y}\sim\mathcal{D}_{\cdot\rightarrow\mathrm{lg}}} \log P_{\theta_d}(\mathbf{y}|\mathbf{x}) \tag{3}$$

where $\mathbf{x}, \mathbf{y}$ is a pair sampled from many-to-one training data.

3.3 Training Algorithm

Algorithm 1 shows the whole training procedure. We will go into the effects of the two-stage design in Section 5. In the first stage, we initialize each module of the Lego-MT model with a pre-trained MT model θ_0. After initialization, we shuffle a one-to-many dataset to obtain a new training sequence for Enc-Flow training. In the second stage, we fix the encoder parameter of M-Flow θ_m and learn the D-Flow decoder θ_d. The iteration keeps running for L epochs. During inference, users can decide to load which flow for inference. We also evaluate the gap between these inference flows in experiments.

4 Experiments

While Lego-MT is generic, we focus the experiments on M2M-100-1.2B as backbone models since M2M-100 is a leading MT model.

4.1 Dataset

Training Data We create a Many-to-Many dataset from OPUS[4]. We build a dataset covering 7 language-specific data and 433 languages. The 7 core languages are En, Zh, De, Ar, Ne, Az, Ceb. The specifics of the construction process are delineated in Appendix A. All training pairs have been deduplicated with *Flores-101*.

Evaluation Data We use *Flores-101* (Fan et al., 2021) as the evaluation set, which provides human-written translation pairs covering 101 languages. Since M2M-100 baselines only cover 86 languages, we only compare Lego-MT with baselines on 86 languages[5]. We evaluate 7×85 translation directions in total.

4.2 Baselines

We conduct experiments by using a pre-trained multilingual machine translation model: M2M-100-1.2B (Fan et al., 2021) as initialization. We build 7 language-specific encoders and 7 language-specific decoders to model 7 core languages. We compare Lego-MT with the following baselines.

Flores-175MB / 615MB *Flores-101* (Goyal et al., 2022) furnishes two baseline models, with parameter sizes of 175MB and 615MB respectively, constructed on M2M-100.

M2M-100-418M It is the smallest model released by Fan et al. (2021), which is a base-version Transformer model with 12 encoders and 12 decoders with 4,096 hidden state units.

M2M-100-1.2B It is a Transformer model released by Fan et al. (2021) with 24 encoders and 24 decoders with 8,192 hidden state units.

M2M-100-12B It is the largest single M2M-100 model released by Fan et al. (2021), which is obtained by adding language-specific layers to M2M-100-1.2B model.

M2M-100-1.2B w. LG-Centric Fine-Tuning To build a fair comparison, we also use the constructed dataset to fine-tune M2M-100-1.2B. We follow the standard fine-tuning paradigm, which uses a Transformer initialized with M2M-100-1.2B. In this baseline, we only use LG-centric data to train models. We simply merge all translation pairs related to language LG together to get the mixed training data. Like our model does, we also add language code in the encoder and decoder parts.

M2M-100-1.2B w. Multilingual Fine-Tuning In order to establish an equitable comparison, the constructed dataset was utilized to fine-tune M2M-100-1.2B. All translation data was amalgamated

[4] https://opus.nlpl.eu/

[5] These 86 languages are: af, am, ar, ast, be, bg, bn, bs, ca, ceb, cs, cy, da, de, el, en, es, et, fa, ff, fi, fr, ga, gl, gu, ha, he, hi, hr, hu, hy, id, ig, is, it, ja, jv, ka, kk, km, kn, ko, lb, lg, ln, lo, lt, lv, mk, ml, mn, mr, ms, my, ne, nl, no, ns, oc, or, pa, pl, ps, pt, ro, ru, sd, sk, sl, so, sr, sv, sw, ta, th, tl, tr, uk, ur, uz, vi, wo, xh, yo, zh, zu.

Model	Param.	X → En	X → Zh	X → De	X → Ar	X → Ne	X → Az	X → Ceb	AVG.
1: Flores-175M (Goyal et al., 2022)	× 0.1	15.7	7.2	11.2	4.6	0.6	3.0	3.1	6.5
2: M2M-100-418M (Fan et al., 2021)	× 0.3	21.2	10.3	14.2	11.5	1.3	2.4	4.9	9.4
3: Flores-615M (Goyal et al., 2022)	× 0.5	21.6	11.0	16.1	8.8	1.0	4.7	5.3	9.8
4: M2M-100-1.2B (Fan et al., 2021)	× 1.0	26.3	12.9	19.3	8.1	1.4	4.6	6.8	11.3
5: M2M-100-12B (Fan et al., 2021)	× 10.0	28.0	13.3	21.3	15.1	2.9	6.4	8.8	13.7
6: (4) + LG-Centric Fine-Tuning	× 1.0	27.9	13.0	19.5	17.2	5.5	4.2	0.5	12.5
7: (4) + Multilingual Fine-Tuning	× 1.0	27.4	13.9	20.9	15.2	12.1	9.4	10.3	15.6
8: Lego-MT	× 1.0	**30.7**	**16.4**	**23.8**	**18.2**	**15.0**	**11.9**	**15.1**	**18.7**
Model	**Param.**	**En → X**	**Zh → X**	**De → X**	**Ar → X**	**Ne → X**	**Az → X**	**Ceb → X**	**AVG.**
1: Flores-175M (Goyal et al., 2022)	× 0.1	12.7	7.8	11.6	6.9	2.2	2.8	5.4	7.1
2: M2M-100-418M (Fan et al., 2021)	× 0.3	17.3	10.1	14.1	11.5	4.0	4.2	6.1	9.6
3: Flores-615M (Goyal et al., 2022)	× 0.5	18.0	11.1	15.6	11.2	5.2	4.3	7.9	10.5
4: M2M-100-1.2B (Fan et al., 2021)	× 1.0	21.5	13.1	17.7	12.6	7.1	6.1	9.5	12.5
5: M2M-100-12B (Fan et al., 2021)	× 10.0	24.7	14.9	20.3	16.4	9.7	6.2	12.5	15.0
6: (4) + LG-Centric Fine-Tuning	× 1.0	21.3	10.9	15.8	14.9	3.9	3.0	1.5	10.2
7: (4) + Multilingual Fine-Tuning	× 1.0	21.8	13.5	18.4	14.7	13.4	11.1	12.4	15.0
8: Lego-MT	× 1.0	**25.0**	**16.3**	**21.4**	**18.4**	**17.0**	**13.5**	**16.8**	**18.3**

Table 1: Translation results on *Flores-101*. The top group shows the results of many-to-one translation and the bottom part shows the results of one-to-many settings. We display the spBLEU on the devtest of *Flores-101*. Each cell represents the average performance of translating from the rest languages. “Param.” represents the number of required parameters during inference. Baseline 7 has the exact same training data with Lego-MT. For a fair comparison, we use Mix-Flow in Lego-MT for all translation pairs. Lego-MT outperforms M2M-100-1.2B w. multilingual fine-tuning by a large margin, with an average gain of 3.2 spBLEU (3.1 on many-to-one translation and 3.3 on one-to-many translation).

for the purpose of fine-tuning M2M-100-1.2B in this baseline. Correspondingly, language codes were incorporated in both the encoder and decoder components, as is done in our model.

4.3 Settings and Metric

Training Details The training code is bulit on the code repository fairseq[6]. Each flow is initialized with a pre-trained M2M-100-1.2B model. We train all models using Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, the learning rate is set to 1e-4, and the max token number is set as 8,000. The training of all centric languages is conducted in random order: En, De, Ne, Az, Ceb, Ar, Zh. We split the whole dataset into 70 shards. And the whole training process takes around 15 days on 32 A100 GPUs.
Metric We use the same evaluation metric (spBLEU) in the *Flores-101* dataset. Before computing BLEU, we de-tokenized all data and then apply sentence piece tokenization for each language. It facilitates a more accurate assessment of model quality on the long-tail of low-resource languages.

4.4 Results

Lego-MT is an efficient translation model, outperforming M2M-100-12B with only 10% inference parameters Table 1 show experiment results on the *Flores-101* devtest set. As we can see, Lego-MT is an efficient translation model that achieves large performance improvements over M2M-100-1.2B, with 7.4 spBLEU improvements on many-to-one translation and 5.8 spBLEU improvements on one-to-many translation. It even outperforms M2M-100-12B especially on one many-to-one settings, with a gain of 5.0 spBLEU. As a comparison, with the same training data, a shared model M2M-100-1.2B only obtains slight performance improvements, 4.3 spBLEU on many-to-one translation, and 2.5 spBLEU on one-to-many translation. These results demonstrate Lego-MT provides an effective solution by using fewer inference parameters to achieve higher results.

Compared with high-resource translation, low-resource translation benefits more from multi-way architectures. We observe that the improvements achieved by Lego-MT are not equally contributed by different languages. As we can see from Table 1, X→Ne, X→Az, and X→Ceb obtain more obvious improvements than X→En, X→Zh, and X→De. On X→Ne translation, Lego-MT even gets 13.6 improvements over M2M-100-1.2B. These results are consistent with previous studies about parameter interference in massive multilingual machine translation that low-resource translation usually suffers. With less parameter interference, Lego-

[6] https://github.com/facebookresearch/fairseq/tree/main/examples/m2m_100.

MT gets higher low-resource translation results.

Multilingual branches play a significant role in avoiding over-fitting. As we can see from Table 1, only fine-tuning M2M-100-1.2B on language-centric data has serious over-fitting problems that the performance is dropped sharply, especially on low-resource settings, with a loss of 3.2 spBLEU on Ne→X translation and 3.1 spBLEU on Az→X translation. Like this baseline, Lego-MT also introduces language-specific parameters but does not show any performance drop. The key difference between Lego-MT with M2M-100-1.2B w. LG-Centric fine-tuning lies in that Lego-MT introduces multilingual branches as regularization, demonstrating that the unified space can avoid catastrophic forgetting.

Lego-MT supports efficient training, which is 28.2× faster than multi-way training. For simplification, we implement an 8-branch architecture where Lego-MT and the multi-way model both have 8 branches for encoder and decoders. We use Chinese-centric data in the first shard and select 7 Zh→X and X→Zh translations as a small training set, which includes high-resource languages (Be, De, Fa, Jv) and low-resource languages (Ne, Pa, Sw). For two models, all 8 encoder branches are initialized with the encoder part of M2M-100-418M, and all 8 decoder branches are initialized with the decoder part of M2M-100-418M. Due to the large parameter size, the multi-way model requires fewer tokens in a single batch. Lego-MT has less parameter size during each inference and thus can support more tokens in a single batch. For a fair comparison, we use the same settings for two models and set the number of tokens in a single batch to 3K. Due to the low GPU efficiency issues, the multi-way model takes 16.9 hours to finish one shard training on average while Lego-MT only takes 0.6 hours.

The total training cost of Lego-MT is only about twice that of M2M-1.2B fine-tuning. In the first stage, we load a multilingual encoder-decoder and a single language-specific encoder, and in the second stage, we load a multilingual encoder-decoder and a single language-specific decoder. Compared to M2M-1.2B, the additional computations come from training language-specific parameters. Since the language-specific branch has the same size as the multilingual branch, the training costs only double. We believe that the training costs for such a model are reasonable, given its one-time training

Method	#Tokens	Size (GB)	Time (Hour)
Multi-Way Training	3,000	60.9	16.9
Lego-MT Training	3,000	10.3	0.6

Table 2: Training efficiency of Lego-MT and a multi-way model. "#Tokens" represent the maximum tokens in a single batch during training. For a fair comparison, we initialize an 8-branch Lego-MT and an 8-branch multi-way model with M2M-100-418M as initialization. Size represents the size of the loaded parameters. "Time" represents the total time of completing all data (We select a small subset of training data for evaluation). In Lego-MT, we use a parallel thread for branch switching, which does not affect the running time. Lego-MT supports efficient training, which achieves 28.2× speedups over multi-way training.

feature. In real-world applications with unlimited data, inference costs are more critical than training costs. The advantage of Lego-MT is that it largely improves translation performance without incurring additional inference costs.

5 Analysis on Lego-MT

Ablation studies on triple-Flow training We design three flows in Lego-MT: Mix-Flow, Enc-Flow, and Dec-Flow. Mix-FLow contains a multilingual encoder and a multilingual decoder, which is essential in regularizing language-specific training. We start from M-Flow and see how Enc-Flow and Dec-Flow affect the final performance, which gives more insights into the design of our framework. For simplification, we use Chinese-centric data in top-10 shards and select 7 Zh→X and X→Zh translation pairs as a small training set, which includes high-resource languages (Be, De, Fa, Jv) and low-resource languages (Ne, Pa, Sw). We train Lego-MT on the selected set and observe results in Table 3. We can see that jointly training Enc-Flow and Mix-Flow boosts the performance in most directions. In contrast, jointly training Dec-Flow and Mix-Flow causes large performance degeneration. It is mainly because that language-specific decoder may cause a large distribution shift on multilingual encoders, resulting in catastrophic forgetting. That's why we split the training into two stages and keeps Dec-Flow in the second stage.

Analysis on inference path section Due to the plug-and-play features, there are several possible inference paths for a single translation direction. At the inference stage, there are three alternative solutions for language-centric translation: Mix-Flow,

Lang	Mix-Flow		Dec-Flow + Mix-Flow		Enc-Flow + Mix-Flow	
	x→zh	zh→x	x→zh	zh→x	x→zh	zh→x
Be	8.1	3.5	7.5	3.3	13.1	6.1
De	16.0	14.7	14.0	13.4	22.1	19.2
Fa	13.1	11.9	11.6	11.3	17.7	15.1
Jv	6.3	3.1	6.0	3.1	8.8	3.3
Ne	9.6	5.6	8.7	4.2	7.2	4.0
Pa	1.4	1.1	1.2	0.5	1.8	0.5
Sw	8.5	8.6	7.5	6.6	12.4	12.4
AVG.	9.0	6.9	8.1	6.1	11.9	8.7

Table 3: Ablation studies on Triple-Flow training. →zh refers to the results of translating to zh and zh→ refers to the results of translating from zh. Dec-Flow brings a large performance drop.

Model	Ceb→Ha	Ceb→Ig	Ceb→Ln	Ceb→Yo	AVG.
M-1.2B	5.5	5.9	0.9	2.4	3.7
M-12B	8.7	10.8	0.9	2.9	5.8
M-FT	6.4	6.6	0.8	2.1	4.0
Lego-MT	**12.5**	**13.9**	**2.3**	**3.2**	**8.0**
Model	**Ha→Ceb**	**Ig→Ceb**	**Ln→Ceb**	**Yo→Ceb**	**AVG.**
M-1.2B	7.4	7.5	4.2	3.4	5.6
M-12B	8.8	8.8	3.8	4.1	6.4
M-FT	7.5	8.6	3.7	4.8	6.2
Lego-MT	**12.3**	**12.3**	**6.2**	**6.7**	**9.4**
Model	**X→Ast**	**X→Da**	**X→Hu**	**X→Lo**	**AVG.**
M-1.2B	**16.7**	22.0	17.7	4.8	15.3
M-12B	13.0	23.3	19.1	**9.0**	16.1
M-FT	13.8	23.2	17.8	0.9	13.9
Lego-MT	15.4	**25.4**	**20.1**	5.6	**16.6**
Model	**Ast→X**	**Da→X**	**Hu→X**	**Lo→X**	**AVG.**
M-1.2B	13.2	18.3	16.0	6.6	13.5
M-12B	15.2	**20.9**	**18.2**	8.8	**15.8**
M-FT	14.4	17.9	15.5	5.8	13.4
Lego-MT	**15.5**	20.8	18.1	**8.9**	**15.8**

Table 4: The results on unseen directions that are not covered by the constructed dataset. "M" means M2M-100. Lego-MT shows the best generalization results. "M-FT" means M2M-100-1.2B w. Multilingual FT.

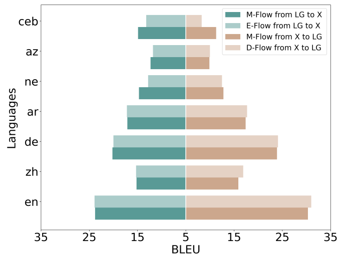

Figure 3: The comparison between different inference paths. For low-resource languages (eg. ceb, az, ne), Mix-Flow (M-encoder + M-decoder) works better than either Enc-Flow (E-encoder + M-decoder) or Dec-Flow (M-encoder + D-decoder). For high-resource languages (eg., en, de, zh, ar), Dec-Flow (a multilingual encoder and a language-specific decoder) achieves better performance among these paths.

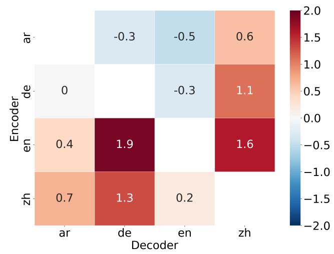

Figure 4: The spBLEU gap between Mix-Flow (a multilingual encoder and a multilingual decoder) and unseen language-specific Flow (the combination of a language-specific encoder and a language-specific decoder). Positive numbers mean the results of the language-specific Flow are better than that of the M-Flow. The unseen language-specific Flow achieves better results on 9 out of 12 directions, demonstrating that Lego-MT can learn the alignment for different branches.

Enc-Flow, and Dec-Flow. Figure 2 shows the comparison between these inference paths. For low-resource languages (eg., Ceb, Az, Ne), Mix-Flow (M-encoder + M-decoder) works better than either Enc-Flow (E-encoder + M-decoder) or Dec-Flow (M-encoder + D-decoder). High-resource languages (eg., En,De,Zh, Ar) prefer language-specific branches. Dec-Flow (a multilingual encoder and a language-specific decoder) achieves better performance among these paths. This demonstrates that specific parameters are more important when the amount of data in a language is huge. In summary, the Mix-Flow (M-encoder + M-decoder) is recommended for inference tasks with low-resource languages, and the Dec-FLow (M-encoder + D-decoder) is more appropriate for high-resource languages.

Lego-MT can learn the align different branches into a unified space. During training, we propose a triple-flow way to train Lego-MT. These three flows contain Mix-Flow, Dec-Flow, and Enc-Flow. To evaluate the quality of the hidden representations, we conduct experiments by directly using a language-specific encoder and a language-specific decoder for inference. Since such combinations do not occur in the training phase, it can evaluate the quality of the unified hidden space. We randomly combine the language-specific encoder and the language-specific decoder of four high-resource languages (En, De, Zh, Ar) with 12 translation directions. Figure 4 shows the performance of directly combining language-specific encoder and decoder. We find that such unseen combinations can get better results in most translation directions (9 out of 12). These results prove that Lego-MT can effectively map all languages into a unified space.

Model	Ast→X	Hu→X	Da→X	Lo→X	En→X	De→X	Ar→X	Az→X	Ceb→X	Ne→X	Zh→X	AVG.
Multilingual FT	3.8	2.6	2.7	6.3	7.6	4.2	1.7	2.6	0.9	6.2	3.7	3.9
Lego-MT	**14.2**	**10.1**	**11.9**	**17.5**	**20.6**	**14.1**	**6.9**	**12.2**	**5.8**	**17.7**	**11.7**	**13.1**
Model	**X→Ast**	**X→Hu**	**X→Da**	**X→Lo**	**X→En**	**X→De**	**X→Ar**	**X→Az**	**X→Ceb**	**X→Ne**	**X→Zh**	**AVG.**
Multilingual FT	4.9	2.2	1.6	6.9	10.8	3.0	0.5	2.7	0.7	7.8	4.3	4.1
Lego-MT	**14.1**	**8.8**	**10.5**	**17.8**	**24.1**	**13.7**	**3.6**	**11.0**	**4.0**	**21.2**	**11.8**	**12.9**

Table 5: Commencing with random initialization, $1/7$ of the data is trained and the performance differential between Multilingual-FT and Lego-MT is evaluated. The findings indicate that the mean performance of Lego-MT across all language orientations surpasses that of Multilingual-FT and exhibits a more rapid convergence rate.

Model	X→En	En→X	AVG.
ChatGPT zero-shot	27.9	23.9	25.9
ChatGPT eight-shot	**31.9**	24.7	**28.3**
Lego-MT	30.2	**25.7**	28.0

Table 6: Comparison of ChatGPT and Lego-MT: zero-shot and eight-shot results. While ChatGPT lags behind Lego-MT in zero-shot performance, it outperforms Lego-MT in the X→En direction with eight-shot. However, in the En→X direction, ChatGPT falls behind Lego-MT even with eight-shot.

In addition, it proves that the performance of high-resource languages still has room for improvement by using language-specific parameters.

Lego-MT achieves promising results in unseen directions. We also conduct experiments on unseen directions to evaluate Lego-MT's performance in these scenarios, as demonstrated in Table 4. Distinguishing unseen translation directions can involve two scenarios: 1) The training data set lacks a specific translation direction. In this case, we start with the low-resource Ceb language and identify translation directions not included in our constructed data set. 2) The training data set lacks a direct translation between two languages. For instance, our training corpus may contain translations from Ast to En and from En to Es, but not a direct translation from Ast to Es. To address this, we randomly select four languages (Ast, Da, Hu, Lo) and evaluate the average performance on the *Flores-101* devtest with one-to-many and many-to-one settings. According to all experimental results, Lego-MT significantly surpasses the Multilingual FT baseline and is on par with the M2M-100-12B.

Lego-MT performance is independent of pre-trained model initialization and converges faster than existing pre-training pipelines. To evaluate the necessity of pre-trained model initialization, we compare Lego-MT with the traditional multilingual pre-training pipeline that uses a single encoder-decoder model for all languages. We conduct experiments on a subset of our constructed corpus, which contains parallel data for 433 languages. We randomly initialize both models and train them on only 1/7 of the data, then measure their performance on *Flores-101*. As shown in Table 5, our experimental results demonstrate that our Lego-MT model is independent of the pre-trained model initialization and achieves faster convergence than the traditional multilingual pre-training pipeline. Moreover, our Lego-MT model outperforms the traditional multilingual pre-training pipeline on most of the machine translation tasks, showing its superior generalization and adaptation ability.

Lego-MT surpasses ChatGPT in the En→X direction and is on par with ChatGPT in the X→En direction, in terms of performance. A comparative analysis between ChatGPT and Lego-MT, as shown in Table 6, reveals that in zero-shot performance, ChatGPT lags behind Lego-MT. However, in eight-shot performance, ChatGPT surpasses Lego-MT in the X→En direction but falls short in the En→X direction. The prompts utilized for ChatGPT are "You are a helpful assistant that translates {*SOURCE_LANG*} to {*TARGET_LANG*}." for the system and "Translate the following {*SOURCE_LANG*} text to {*TARGET_LANG*}: {*SOURCE_TEXT*}." for the user.

6 Conclusion

With the increasing scale of languages, using a single model to translate all directions brings new challenges in practice. This paper proposes an efficient training recipe, which results in a detachable multilingual translation model, Lego-MT. To validate the effectiveness of our algorithm, we develop a massive MNMT translation dataset, covering 433 languages. Results on *Flores-101* show that Lego-MT-1.2B achieves large performance improvements over strong baselines under a fair comparison. It even outperforms the result of M2M-12B with a gain of 4 BLEU on many-to-one.

References

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. *arXiv preprint arXiv:1903.00089*.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*.

Ankur Bapna and Orhan Firat. 2019. Simple, scalable adaptation for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1538–1548, Hong Kong, China. Association for Computational Linguistics.

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, Loïc 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. *CoRR*, abs/2207.04672.

Raj Dabre, Chenhui Chu, and Anoop Kunchukuttan. 2020. A survey of multilingual neural machine translation. *ACM Comput. Surv.*, 53(5).

Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Stablemoe: Stable routing strategy for mixture of experts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 7085–7095. Association for Computational Linguistics.

Siddharth Dalmia, Dmytro Okhonko, Mike Lewis, Sergey Edunov, Shinji Watanabe, Florian Metze, Luke Zettlemoyer, and Abdelrahman Mohamed. 2022. Legonn: Building modular encoder-decoder models.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. 2015. Multi-task learning for multiple language translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics(ACL)*.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pages 5547–5569. PMLR.

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, et al. 2021. Beyond english-centric multilingual machine translation. *J. Mach. Learn. Res.*, 22(107):1–48.

William Fedus, Barret Zoph, and Noam Shazeer. 2021. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzman, and Angela Fan. 2022. The flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10:522–538.

Jiatao Gu, Hany Hassan, Jacob Devlin, and Victor O.K. Li. 2018. Universal neural machine translation for extremely low resource languages. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*.

Thanh-Le Ha, Jan Niehues, and Alex Waibel. 2016a. Toward multilingual neural machine translation with universal encoder and decoder. In *Proceedings of the 13th International Conference on Spoken Language Translation*, Seattle, Washington D.C. International Workshop on Spoken Language Translation.

Thanh-Le Ha, Jan Niehues, and Alex Waibel. 2016b. Toward multilingual neural machine translation with universal encoder and decoder. In *Proceedings of the 13th International Conference on Spoken Language Translation*, Seattle, Washington D.C. International Workshop on Spoken Language Translation.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Baijun Ji, Zhirui Zhang, Xiangyu Duan, Min Zhang, Boxing Chen, and Weihua Luo. 2020. Cross-lingual pre-training based transfer for zero-shot neural machine translation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 115–122.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation.

Transactions of the Association for Computational Linguistics, 5:339–351.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully character-level neural machine translation without explicit segmentation. *Transactions of the Association for Computational Linguistics*, 5:365–378.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

Xian Li and Hongyu Gong. 2021. Robust optimization for multilingual translation with imbalanced data. *Advances in Neural Information Processing Systems (NeurIPS)*.

Zehui Lin, Xiao Pan, Mingxuan Wang, Xipeng Qiu, Jiangtao Feng, Hao Zhou, and Lei Li. 2020. Pre-training multilingual neural machine translation by leveraging alignment information. In *the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Zehui Lin, Liwei Wu, Mingxuan Wang, and Lei Li. 2021. Learning language specific sub-network for multilingual machine translation. In *the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

Sungwon Lyu, Bokyung Son, Kichang Yang, and Jaekyoung Bae. 2020. Revisiting Modularized Multilingual NMT to Meet Industrial Demands. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5905–5918, Online. Association for Computational Linguistics.

Graham Neubig and Junjie Hu. 2018. Rapid adaptation of neural machine translation to new languages. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Xiao Pan, Liwei Wu, Mingxuan Wang, and Lei Li. 2021. Contrastive learning for many-to-many multilingual neural machine translation. In *the 59th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Devendra Sachan and Graham Neubig. 2018. Parameter sharing methods for multilingual self-attentional translation models. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 261–271, Brussels, Belgium. Association for Computational Linguistics.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Zhenqiao Song, Hao Zhou, Lihua Qian, Jingjing Xu, Shanbo Cheng, Mingxuan Wang, and Lei Li. 2021. switch-glat: Multilingual parallel machine translation via code-switch decoder. In *International Conference on Learning Representations (ICLR)*.

Zewei Sun, Mingxuan Wang, and Lei Li. 2021. Multilingual translation via grafting pre-trained language models. In *the Conference on Empirical Methods in Natural Language Processing (EMNLP) - Findings*.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. 2020. Improving massively multilingual neural machine translation and zero-shot translation. *arXiv preprint arXiv:2004.11867*.

Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. 2021. Counter-interference adapter for multilingual machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 2812–2823.

Barret Zoph and Kevin Knight. 2016. Multi-source neural translation. *arXiv preprint arXiv:1601.00710*.

Limitation

Despite promising results, we also notice several limitations in this paper. First, we find that low-resource translation is not boosted by language-specific decoders and language-specific encoders, which require more exploration of the trade-off between parameter sharing and parameter tension. Second, the evaluation of few-shot languages still remains a large problem. Although the final training dataset covers 433 languages, we only evaluate the translation performance on the available evaluation set that only covers 86 languages since baselines do not support so many languages. More standard benchmarks are required for evaluation.

A Dataset construction

In this section, we will describe the construction details of the Many-to-Many dataset. As shown in Table 5, the pipeline mainly consists of six steps:

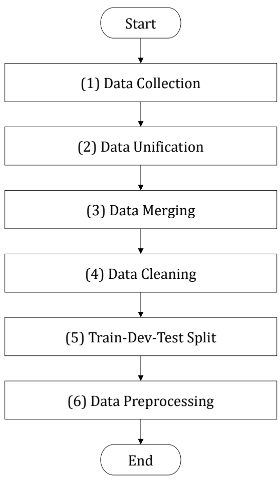

Figure 5: The construction pipeline for Many-to-Many dataset.

Step 1: Data Collection The raw data is collected from OPUS[7], which is an open corpus that collects numerous parallel sentences from the web and overs a large number of domains from legislative to religious texts.

Step 2: Data Unification Since the OPUS includes datasets from different sources, it leads to the following two significant issues.

1) Different Language Code: Some language in OPUS has several corresponding language codes. One of the reasons is that different corpora use different standards for language code, including ISO 639-1, ISO 639-2, ISO 639-3 or self-defined language codes in OPUS. Another reason is that some corpora append region ids at the end of language codes to distinguish the same language used in different regions. To unify language codes, we replace ISO 639-2 and ISO 639-3 language codes with ISO 639-1 language codes if the codes from ISO 639-1, ISO 639-2 and ISO 639-3 have the same language name in the code set published by SIL International (formerly known as the Summer Institute of Linguistics)[8].

2) Inconsistent Operation: Some datasets pre-tokenize their sentences in OPUS, especially for Chinese and Japanese.

Therefore, we remove the region id if the language code ends with a region id. All replaced language codes are shown in Table 7. For the language codes out of ISO 639 series, we list them and the corpus they come from in Table 9. Furthermore, we report all used language codes and the full names of their corresponding languages in our dataset in Table 10. Then we detokenize all sentences by removing white space and unifying our texts.

Step 3: Data Merging After data unification, the parallel data is merged with the same language code pair from a different corpus.

Step 4: Data Cleaning The OPUS corpus collected from the web contains some poor-quality data. The main problems are:

1) Duplication: we use the deduplication script from fairseq[9] to remove all duplicated sentence pairs for each language pair.

2) Missing Translation: We remove the sentence without corresponding translation or repeating itself as translation.

3) Length Mismatching: After segmentation the sentences with white space for most languages or individual characters for Chinese and Japanese, we apply a filtering script from Moses decoder[10] to remove the sentences that the length is more than 250 words or three times difference in length between source and target sentences.

[7] https://opus.nlpl.eu/

[8] https://iso639-3.sil.org/sites/iso639-3/files/downloads/iso-639-3.tab

[9] https://github.com/facebookresearch/fairseq/edit/main/examples/backtranslation/deduplicate_lines.py

[10] https://github.com/moses-smt/mosesdecoder

Step 5: Train-Dev-Test Split Different train-dev-test split schemes are developed based on the data quantity.

1) A parallel data with more than 6,000 sentence pairs. We randomly sample separately about 2,000 sentence pairs as validation and test set, respectively. And the rest is train set.

2) A parallel data with fewer than 6,000 sentence pairs. We take 80%, 10%, 10% of all samples as train, validation, and test.

To avoid any overlap between our training data and used benchmark test data, we filter all sentences that exist in the common benchmarks (WMT, *Flores-101*) from our train and validation set.

Step 6: Data Preprocessing The data preprocessing consists of two main steps:

1) Sampling: Because the full dataset is huge, we sample some data for our training. The final dataset contains 1,307,143,514 sentence pairs, 433 languages, and 1,922 training pairs.

2) Preprocessing: The data is preprocess using the SentencePiece tokenizer provided by Fan et al. (2021) with a shared vocabulary of size 128,112.

Original	Replaced	Original	Replaced	Original	Replaced
ak	aka	es	es_HN	pt	pt_BR
am	amh	es	es_EC	pt	pt_br
ar	ara	es	es_CO	pt	pt_PT
ar	ar_SY	fa	fa_IR	rn	run
ar	ar_TN	fa	fa_AF	rw	kin
ay	aym	ff	ful	sn	sna
az	az_IR	fr	fr_FR	so	som
bg	bg_BG	fr	fr_CA	sr	srp
bm	bam	fr	fr_BE	sr	sr_ME
bn	bn_IN	fr	fr_ca	st	sot
ca	cat	ha	hau	sw	swa
da	da_DK	hi	hi_IN	ta	ta_LK
de	de_CH	ig	ibo	tg	tg_TJ
de	de_AT	it	it_IT	ti	tir
de	de_DE	jp	jap	tl	tl_PH
es	es_CL	kr	kau	tr	tr_TR
es	es_SV	kv	kpv	ur	ur_PK
es	es_NI	ln	lin	vi	vi_VN
es	es_UY	mg	mlg	wo	wol
es	es_PE	ms	ms_MY	xh	xho
es	es_VE	nb	nb_NO	yo	yor
es	es_AR	nds	nds_nl	ze	ze_zh
es	es_MX	nl	nl_NL	ze	ze_en
es	es_PA	nl	nl_BE	zh	zh_cn
es	es_CR	nn	nn_NO	zh	zh_CN
es	es_PR	no	no_nb	zhtrad	zh_HK
es	es_ES	ny	nya	zhtrad	zh_TW
es	es_GT	om	orm	zhtrad	zh_tw
es	es_DO	pa	pan	zu	zul

Table 7: Code Replacement List. We use the codes in the column "Original" to replace the codes in the column "replaced" if these replaced codes exist in OPUS.

Code	En	De	Ar	Zh	Ne	Az	Ceb
#Sentence Pairs	811,238,712	360,369,144	152,457,830	92,763,445	6,654,270	4,208,025	1,683,531

Table 8: The number of sentence pairs for each core language in Lego-MT training,

Code	Dataset	Code	Dataset	Code	Dataset	Code	Dataset	Code	Dataset
crp	bible-uedin	cb	MultiCCAligned	sz	MultiCCAligned	sgn	QED	cycl	Tatoeba
tc	EUbookshop	cx	MultiCCAligned	zz	MultiCCAligned	iro	QED	nah	Tatoeba
zhs	GlobalVoices	ns	MultiCCAligned	ze	OpenSubtitles	mo	QED,Ubuntu		
zht	GlobalVoices	qd	MultiCCAligned	bh	QED	ber	QED,Ubuntu		
tmp	GNOME	qa	MultiCCAligned	bnt	QED	toki	Tatoeba		
gr	GNOME	tz	MultiCCAligned	ry	QED	kzj	Tatoeba		

Table 9: Unkown Language Codes, which are out of ISO 639 series. We can't confirm their full names.

Language	Code	Language	Code	Language	Code	Language	Code	Language	Code	Language	Code
Abkhazian	ab	Corsican	co	Iban	iba	Lower Sorbian	dsb	Ossetian	os	Swahili (macrolanguage)	sw
Achinese	ace	Cree	cr	Icelandic	is	Lukpa	dop	Ottoman Turkish (1500-1928)	ota	Swati	ss
Achuar-Shiwiar	acu	Creek	mus	Ido	io	Luo (Kenya and Tanzania)	luo	Paite Chin	pck	Swedish	sv
Adyghe	ady	Crimean Tatar	crh	Igbo	ig	Lushootseed	lut	Palauan	pau	Swiss German	gsw
Afar	aa	Croatian	hr	Iloko	ilo	Luxembourgish	lb	Pali	pi	Syriac	syr
Afrihili	afh	Cusco Quechua	quz	Indonesian	id	Luyia	luy	Pampanga	pam	Tachawit	shy
Afrikaans	af	Czech	cs	Ingrian	izh	Macedonian	mk	Pangasinan	pag	Tachelhit	shi
Aguaruna	agr	Danish	da	Ingush	inh	Macedo-Romanian	rup	Panjabi	pa	Tagal Murut	mvv
Ainu (Japan)	ain	Dari	prs	Interlingua	ia	Madurese	mad	Papiamento	pap	Tagalog	tl
Akan	ak	Dinka	din	Interlingue	ie	Maithili	mai	Papuan Malay	pmy	Tahaggart Tamahaq	thv
Akawaio	ake	Drents	drt	Inuktitut	iu	Malagasy	mg	Pedi	nso	Tahitian	ty
Aklanon	akl	Dungan	dng	Inupiaq	ik	Malay (individual language)	zlm	Pennsylvania German	pdc	Tajik	tg
Albanian	sq	Dutch	nl	Iranian Persian	pes	Malay (macrolanguage)	ms	Persian	fa	Talossan	tzl
Algerian Arabic	arq	Dutton World Speedwords	dws	Irish	ga	Malayalam	ml	Phoenician	phn	Talysh	tly
American Sign Language	ase	Dzongkha	dz	Italian	it	Maltese	mt	Picard	pcd	Tamashek	tmh
Amharic	am	Eastern Canadian Inuktitut	ike	Jakun	jak	Mam	mam	Piemontese	pms	Tamil	ta
Ancient Greek (to 1453)	grc	Eastern Mari	mhr	Jamaican Creole English	jam	Mambae	mgm	Pipil	ppl	Tarifit	rif
Ancient Hebrew	hbo	Eastern Maroon Creole	djk	Japanese	ja	Mandarin Chinese	cmn	Plateau Malagasy	plt	Tase Naga	nst
Arabic	ar	Efik	efi	Javanese	jv	Manx	gv	Polish	pl	Tatar	tt
Aragonese	an	Egyptian Arabic	arz	Jewish Babylonian Aramaic	tmr	Maori	mi	Portuguese	pt	Telugu	te
Armenian	hy	Emilian	egl	Kabyle	kab	Marathi	mr	Potawatomi	pot	Tena Lowland Quichua	quw
Arpitan	frp	English	en	Kadazan Dusun	dtp	Marshallese	mh	Prussian	prg	Tetelcingo Nahuatl	nhg
Asháninka	cni	Erzya	myv	Kalaallisut	kl	Mesopotamian Arabic	acm	Pushto	ps	Tetum	tet
Assamese	as	Esperanto	eo	Kalmyk	xal	Miahuatlán Zapotec	zam	Quechua	qu	Thai	th
Asturian	ast	Estonian	et	Kamba (Kenya)	kam	Middle English (1100-1500)	enm	Quenya	qya	Tibetan	bo
Avaric	av	Evenki	evn	Kannada	kn	Middle French (ca. 1400-1600)	frm	Quiotepec Chinantec	chq	Tigrinya	ti
Avestan	ae	Ewe	ee	Kanuri	kr	Mikasuki	mik	Rapanui	rap	Tohono O'odham	ood
Awadhi	awa	Extremaduran	ext	Kaqchikel	cak	Mi'kmaq	mic	Romanian	ro	Tok Pisin	tpi
Aymara	ay	Faroese	fo	Karelian	krl	Min Dong Chinese	cdo	Romansh	rm	Tonga (Tonga Islands)	to
Azerbaijani	az	Fiji Hindi	hif	Kashmiri	ks	Min Nan Chinese	nan	Romany	rom	Traditional Chinese	zhtrad
Baluchi	bal	Fijian	fj	Kashubian	csb	Minangkabau	min	Rundi	rn	Tsonga	ts
Bambara	bm	Filipino	fil	Kazakh	kk	Mingrelian	xmf	Russian	ru	Tswana	tn
Banjar	bjn	Finnish	fi	Kekchí	kek	Mirandese	mwl	Rusyn	rue	Tupí	tpw
Barasana-Eduria	bsn	French	fr	Khakas	kjh	Místkito	miq	Samoan	sm	Turkish	tr
Bashkir	ba	Friulian	fur	Khasi	kha	Modern Greek (1453-)	el	Samogitian	sgs	Turkmen	tk
Basque	eu	Fulah	ff	Khmer	km	Mohawk	moh	Sango	sg	Tuvalu	tvl
Bavarian	bar	Galela	gbi	K'iche'	quc	Mongolian	mn	Sanskrit	sa	Twi	tw
Baybayanon	bvy	Galician	gl	Kikuyu	kik	Morisyen	mfe	Santali	sat	Uab Meto	aoz
Belarusian	be	Gan Chinese	gan	Kinyarwanda	rw	Moroccan Arabic	ary	Sardinian	sc	Udmurt	udm
Bemba (Zambia)	bem	Ganda	lg	Kirghiz	ky	Mossi	mos	Saterfriesisch	stq	Uighur	ug
Bengali	bn	Garhwali	gbm	Klingon	tlh	Nauru	na	Scots	sco	Ukrainian	uk
Berom	bom	Georgian	ka	Koasati	cku	Navajo	nv	Scottish Gaelic	gd	Uma	ppk
Bhojpuri	bho	German	de	Kölsch	ksh	Neapolitan	nap	Sediq	trv	Umbundu	umb
Bislama	bi	Gheg Albanian	aln	Komi	kv	Nepali (individual language)	npi	Serbian	sr	Upper Sorbian	hsb
Bodo (India)	brx	Gilbertese	gil	Komi-Permyak	koi	Nepali (macrolanguage)	ne	Serbo-Croatian	sh	Urdu	ur
Bosnian	bs	Goan Konkani	gom	Kongo	kg	Nigerian Fulfulde	fuv	Shan	shn	Uspanteco	usp
Breton	br	Gothic	got	Korean	ko	Niuean	niu	Shona	sn	Uzbek	uz
Brithenig	bzt	Gronings	gos	Kotava	avk	Nogai	nog	Shuar	jiv	Venda	ve
Buginese	bug	Guadeloupean Creole French	gcf	Kriang	ngt	North Levantine Arabic	apc	Shuswap	shs	Venetian	vec
Bulgarian	bg	Guarani	gn	Kuanyama	kj	North Moluccan Malay	max	Sicilian	scn	Vietnamese	vi
Buriat	bua	Guerrero Amuzgo	amu	Kurdish	ku	Northern Frisian	frr	Silesian	szl	Vlaams	vls
Burmese	my	Guerrero Nahuatl	ngu	Kven Finnish	fkv	Northern Kurdish	kmr	Sindarin	sjn	Volapük	vo
Cabécar	cjp	Gujarati	gu	Láadan	ldn	Northern Sami	se	Sindhi	sd	Walloon	wa
Camsá	kbh	Gulf Arabic	afb	Ladin	lld	Northwestern Ojibwa	ojb	Sinhala	si	Walser	wae
Catalan	ca	Haida	hai	Ladino	lad	Norwegian	no	Slovak	sk	Waray (Philippines)	war
Cebuano	ceb	Haitian	ht	Lakota	lkt	Norwegian Bokmål	nb	Slovenian	sl	Welsh	cy
Central Huasteca Nahuatl	nch	Hakha Chin	cnh	Lao	lo	Norwegian Nynorsk	nn	Somali	so	Western Frisian	fy
Central Kurdish	ckb	Hakka Chinese	hak	Latgalian	ltg	Novial	nov	South Azerbaijani	azb	Western Panjabi	pnb
Central Sama	sml	Hausa	ha	Latin	la	Nuer	nus	South Ndebele	nr	Wolaytta	wal
Chamorro	ch	Hawaiian	haw	Latvian	lv	Nyanja	ny	Southern Kurdish	sdh	Wolof	wo
Chavacano	cbk	Hebrew	he	Ligurian	lij	Occitan (post 1500)	oc	Southern Sami	sma	Wu Chinese	wuu
Chechen	ce	Hiligaynon	hil	Limburgan	li	Old English (ca. 450-1100)	ang	Southern Sotho	st	Xhosa	xh
Cherokee	chr	Hindi	hi	Lingala	ln	Old French (842-ca. 1400)	fro	Southwestern Dinka	dik	Yakut	sah
Chhattisgarhi	hne	Hiri Motu	ho	Lingua Franca Nova	lfn	Old Frisian	ofs	Spanish	es	Yaqui	yaq
Chinese	zh	Hmong Daw	mww	Literary Chinese	lzh	Old Norse	non	Standard Malay	zsm	Yiddish	yi
Choctaw	cho	Ho	hoc	Lithuanian	lt	Old Russian	orv	Standard Moroccan Tamazight	zgh	Yoruba	yo
Church Slavic	cu	Huastec	hus	Liv	liv	Old Spanish	osp	Sumerian	sux	Zarma	dje
Chuvash	cv	Hungarian	hu	Lojban	jbo	Oriya (macrolanguage)	or	Sundanese	su	Zaza	zza
Coptic	cop	Hunsrik	hrx	Lombard	lmo	Orizaba Nahuatl	nlv	Swabian	swg	Zulu	zu
Cornish	kw	Hupa	hup	Low German	nds	Oromo	om	Swahili (individual language)	swh		

Table 10: List of Languages. Our dataset mainly use ISO 639 series as language code. For traditional Chinese, we define "zhtrad" as code.