Efficiently Upgrading Multilingual Machine Translation Models to Support More Languages

Simeng Sun[1*] **Maha Elbayad**[2] **Anna Sun**[2] **James Cross**[2†]

University of Massachusetts Amherst[1]   Meta AI[2]

simengsun@umass.edu

{elbayadm, annaysun, jcross}@meta.com

Abstract

With multilingual machine translation (MMT) models continuing to grow in size and number of supported languages, it is natural to reuse and upgrade existing models to save computation as data becomes available in more languages. However, adding new languages requires updating the vocabulary, which complicates the reuse of embeddings. The question of how to reuse existing models while also making architectural changes to provide capacity for both old and new languages has also not been closely studied. In this work, we introduce three techniques that help speed up effective learning of the new languages and alleviate catastrophic forgetting despite vocabulary and architecture mismatches. Our results show that by (1) carefully initializing the network, (2) applying learning rate scaling, and (3) performing data up-sampling, it is possible to exceed the performance of a same-sized baseline model with 30% computation and recover the performance of a larger model trained from scratch with over 50% reduction in computation. Furthermore, our analysis reveals that the introduced techniques help learn the new directions more effectively and alleviate catastrophic forgetting at the same time. We hope our work will guide research into more efficient approaches to growing languages for these MMT models and ultimately maximize the reuse of existing models.

1 Introduction

Research into multilingual machine translation (MMT) (Aharoni et al., 2019; Fan et al., 2020) has shifted from a relatively small number of translation directions (Dong et al., 2015; Firat et al., 2016; Ha et al., 2016) to much larger scale, recently reaching up to tens of thousands of translation directions (NLLB Team et al., 2022; Bapna et al., 2022). Despite the notable increase in the number of supported languages, these models still need to be upgraded as increasing amount of data in new languages are becoming available. The process of adding new languages to existing models is an instance of continual learning (Ring, 1994), in which the models need to effectively learn new tasks (new translation directions) while not catastrophically forgetting (French, 1993; McCloskey and Cohen, 1989) the knowledge about tasks from the previous training stage (original translation directions).

Unlike the most conventional continual learning setup where the model remains the same as the initial learning stage, growing languages for MMT models needs to deal with *new* parameters. Adding new languages to existing training data shifts sub-word token distribution (e.g., tokens from unseen scripts are added), hence the need to re-train the tokenizer, which adds new embedding parameters to the MMT model. Previous studies have shown the effectiveness of adapting embedding parameters to retain performance on old translation directions (Lakew et al., 2018; Escolano et al., 2019; Garcia et al., 2021). This is usually done by reusing the embeddings for tokens that overlap between the old and new vocabularies.

One aspect that has not been extensively addressed in previous research is how to deal with other architecture mismatches that may arise during the continual learning phase. When growing MMT models to support many additional languages, it also makes sense to increase the model size overall. This extra capacity can be used to learn not only the new directions well but also the old directions better. It is not obvious, however, how to reuse the parameters from existing models (i.e., to engage in continual learning) given such architectural changes. Thus, in addition to dealing with different vocabularies, we also investigate how to make use of previously trained models when scaling up model size in order to train much more efficiently than training from scratch.
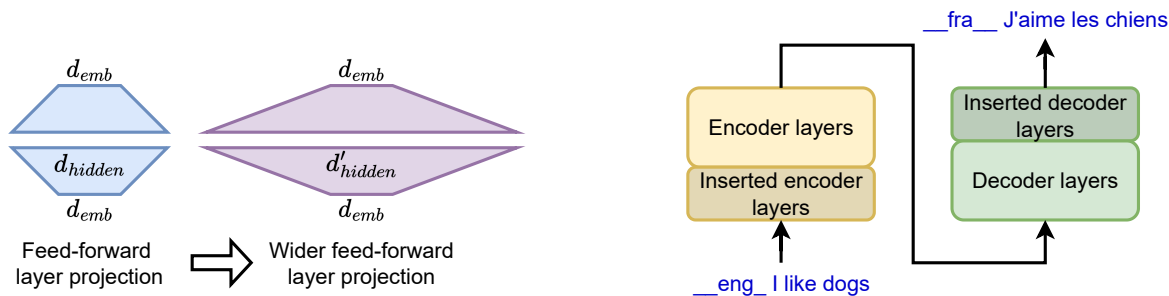

Figure 1: Illustration of two architectural mismatches we tackle in this paper. **Left:** The hidden dimension in the feed-forward layers is doubled ($d'_{\text{hidden}} = 2d_{\text{hidden}}$) during the continual learning stage so that the model becomes "wider". **Right:** Additional layers are inserted at the bottom of encoder and the top of decoder so that the model becomes "deeper". Both architectural changes increase capacity and are not well-addressed by previous works.

In this work, we introduce a recipe that helps significantly reduce the amount of required computation for continually learning new languages in MMT models. The recipe involves training the models with a combination of three techniques: (1) careful initialization of the network (2) applying learning rate scaling and (3) up-sampling selected language pairs. We validate our method on settings both with and without *architecture mismatches* (e.g., models becoming wider or deeper as shown in Figure 1 to have extra capacity for both old and new directions). We compare our models with strong same-sized baselines that are trained on all data from scratch. Our experimental results show that, without architecture mismatches, it is possible to outperform the baseline with just 30% of computation required by the baseline. When training larger models, less than 50% of the original computation is required to match the full performance of the wider baseline model, and less than 10% of computation is needed to recover over 95% of the corresponding baseline performance. We further conduct a suite of analysis which shows that:

- Proper initialization of the parameters before continual learning is crucial for fast convergence.

- Data up-sampling is vital to achieving good performance on new language pairs.

- Scaling down the learning rate for reused parameters helps alleviate catastrophic forgetting.

It is our hope that this work will help save computation for future research into large-scale multilingual machine translation, guide more efficient reuse

of existing models for continual learning, and allow people to efficiently adapt large publicly-released MMT models for new languages and datasets.

2   Method

Adding new languages to existing models, especially languages in new scripts, leads to different subword tokenization, thus different vocabularies, which precludes simple fine-tuning with the exact same model on additional data. During the continual learning stage, we may also want to increase the model size overall to have extra capacity to learn the new languages and improve old languages at the same time. Therefore, we also investigate two typical architectural changes commonly done to increase the network capacity: (1) make the model "wider" by expanding the hidden dimension of the feed-forward layers, and (2) make the model "deeper" by inserting new layers to both encoder and decoder. In this section, we delineate three techniques that we found most effective in achieving computation reduction for the continual learning of MMT models.

Proper initialization. While we can copy weights from the old model[1] to the new model, it is not immediately clear how the new parameters (e.g., new token embeddings, new feed-forward weights, new layers) should be initialized and co-adapted with the old weights such that maximal knowledge about the old directions is retained. Instead of initializing the new parameters randomly, we find that initializing the new embeddings with that of <unk> leads to the best performance. When

---

[1]The weights that can be directly copied without any modification include (1) the token embeddings that overlap between old and new model, (2) same-shaped non-embedding parameters.

		M20	**M25**[@100k]	**Mt25**[@30k]	**M25**$_{wide}^{@100k}$	**Mt25**$_{wide}^{@50k}$	**M25**$_{deep}^{@100k}$	**Mt25**$_{deep}^{@50k}$
spBLEU	All	33.4	31.6	31.8	32.8	32.8	32.8	32.4
	Orig.	33.4	33.4	33.5	34.5	34.25	34.5	33.9
	Added	-	24.6	25.2	25.9	27.2	26.2	26.2
chrF++	All	76	74	74	75	75	75	75
	Orig.	76	76	76	76	76	76	76
	Added	-	67	69	68	70	68	69

Table 1: While continually learning new languages by bootstrapping from a model trained on 20 languages (M20), given new embedding parameters (vocabulary mismatch), one can exceed the performance of a baseline model trained on all languages from scratch (M25[@100k]) with just 30% computation (Mt25[@30k]). With architectural mismatches, employing our method, the wider-model baseline performance (M25$_{wide}$) and over 98% performance of a deeper-model baseline (M25$_{deep}^{@100k}$) can be recovered with half of the corresponding baseline computation (Mt25$_{wide}^{@50k}$ and Mt25$_{deep}^{@50k}$ respectively). The "Orig." row shows performance on old 20 languages and the "Added" on newly added 5 languages.

the network becomes "wider" in the continual learning phase, concatenating each old weight matrix with a noisy version of itself performs better than other methods we tried. When the network becomes "deeper", initializing new layers with averaged weights of old layers results in slight improvement over other naive initialization methods. For each setup, we compare different initialization methods in section 4.2.

Data up-sampling. Similar to (Garcia et al., 2021), we introduce the new tasks by mixing old and new training data together. Since the main goal of the continual learning phase is to quickly learn the new directions, we up-sample the new pairs so the model gets more learning signals from these new directions. To increase the transfer across related language pairs and reduce the chance of catastrophically forgetting less represented directions in the original training data, we also up-sample a small number of the old low-resource languages that are from the same language family as the new languages. We present the effect of up-sampling these selected directions in section 4.3.

Learning rate scaling. To further mitigate the frequent catastrophic forgetting problem exhibited in continual learning (Ring, 1994; Thompson et al., 2019), we scale the learning rate for individual parameters depending on whether they are copied from the old model or not. Based on the assumption that the model better retains the knowledge about old tasks if the weights stay close to that of the old model (Kirkpatrick et al., 2017), we scale down the learning rate for these old parameters while maintaining or scaling up the learning rate for the newly

added parameters. In contrast to other methods that incur extra computation such as Fisher Information based loss (Thompson et al., 2019), our approach is simple, straightforward and efficient in alleviating catastrophic forgetting. We present a deeper analysis of learning rate scaling in section 4.4 and section 5.2.

3 Experiments

Languages. We conduct our experiments on 25 languages covering ten language families, four resource levels (high, mid, low, and very low) and four scripts[2]. To grow the languages atop existing learned languages, we train the seed model on 20 languages (40 English-centric directions) and add 5 new languages (10 English-centric directions) during the continual learning stage. Mimicking the common scenario where newly added languages are mostly low-resource, we select five low and very-low resource languages covering four language families and four scripts as the new languages to add to the seed model. To verify the validity of our approach, we also experiment with two other 20/5 groupings and one 12/13 division of old/new languages in section 5.1[3]. Table 2 shows the groups of original and added languages used in most of our experiments. For all our models, we train sentencepiece (Kudo, 2018)[4] tokenizers with 64K tokens and a sampling temperature of 2, on joined (source and target) data from all language directions available in each setup. This

---

[2]We include more details about these languages in Appendix A Table 9.

[3]More language grouping details are shown in Table 10 and Table 11 in Appendix A.

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

	Original				Added
lav	rus	fin	spa		xho
lit	hin	est	deu		guj
swh	mar	pol	zul*		npi
ukr	mkd	ces	msa*		ind
bel	bul	fra	kir*		kaz

Table 2: The M20 model is trained on the "Original" languages. Mt25, Mt25$_{wide}$ and Mt25$_{deep}$ bootstrap from M20 and train on the combination of "Original" and "Added" languages. We perform data up-sampling over added data in conjunction with related old low-resource languages (marked with *).

means that tokenizers in the initial training on the the old languages, were not exposed to unseen languages. The data we used are subsets of data used by NLLB Team et al. (2022) without any mined or back-translated examples.

Models. Our baseline models have a standard 12-layer encoder and 12-layer decoder Transformer architecture (Vaswani et al., 2017) with 16 attention heads, embedding dimension 1024 and hidden dimension 4096. We refer to the baseline model trained on 20 languages and 25 languages from scratch as M20 and M25 respectively. For wider models, we double the hidden dimension size to 8192; and for deeper models, we insert 6 extra layers to encoder and decoder each. We refer to the baseline wide and deep models trained on 25 languages from scratch as M25$_{wide}$ and M25$_{deep}$ respectively, and refer to the models which bootstrap from the M20 model and trained to support new languages while remaining the same architecture, using a wider network, and deeper network as Mt25, Mt25$_{wide}$ and Mt25$_{deep}$ respectively. All translation directions were evaluated on FloRes (Goyal et al., 2022; Guzmán et al., 2019) dev set and use spBLEU[5] dand chrF++ (Popović, 2017) to evaluate the best checkpoint selected by validation loss with beam size 4.[6]

Training details We train baseline models (M25, M25$_{wide}$, M25$_{deep}$) for 100K updates with batch

[6]We use these two metrics instead of BLEU or pre-trained metrics (Kocmi et al., 2021) because over half of the languages we use are of mid- to low-resource and Goyal et al. (2022) show spBLEU treats low-resource languages more fairly.

	All	Orig.	Added
Mt25	**31.8**	**33.5**	**25.2**
random init all	28.2*	29.4*	23.4*
random init new	31.7	33.5	24.9*
no up-sampling	31.3*	33.4	22.6*
no lr scaling	31.5*	33.1*	25.1
Mt25$_{wide}$	**32.6**	**34.1**	**26.3**
random init all	26.7*	28.0*	21.5*
random init new	29.9*	31.3*	24.6*
no up-sampling	31.9*	34.1	23.3*
no lr scaling	32.3*	33.9*	26.2*
Mt25$_{deep}$	**32.2**	33.8	25.6
random init all	30.5*	31.7*	**25.7**
random init new	32.2	**33.9**	25.6
no up-sampling	31.2*	33.5*	22.2*
no lr scaling	31.6*	33.0*	26.1

Table 3: Ablation of the effect of not using M20 weights (random init all), the effect of not carefully initializing new parameters (random init new), not having data up-sampling and not applying learning rate scaling across three scenarios. All numbers are spBLEU scores of models evaluated after 30K updates. * indicates p-value of paired T-test against corresponding best model (Mt25, Mt25$_{wide}$, Mt25$_{deep}$) is smaller than 0.05.

size $\sim$444K tokens peak learning rate 0.003 warmed up with 8000 steps. For the bootstrapped models, we train Mt25 for 30k updates and Mt25$_{wide}$ and Mt25$_{deep}$ for 50k updates since there are more added parameters. We use temperature 1 and prepend encoder/decoder language token at the beginning of each example. All models are trained with attention dropout 0.1 and label smoothing with $\epsilon = 0.1$ (Szegedy et al., 2016). The baseline models M25, M25$_{wide}$ and M25$_{deep}$ complete training of 100K updates on 64 GPUs in $\sim$24h, $\sim$29.9h, and $\sim$29.0h respectively. We do not reset the learning rate scheduler for the second training phase.[7] The data of selected languages are up-sampled by 5 and the learning rate of the old parameters in Mt25$_{wide}$ is multiplied by 0.5 and in Mt25$_{deep}$ by 0.05 at the beginning while linearly increasing to 0.5. We present the effect of these hyperparameters in the next section.

[7]We tried resetting learning scheduler (i.e., learning rate warms up 8K steps from 0 to 0.003 and then starts to decrease), small constant learning rate, and not resetting learning rate scheduler in our preliminary experiments and the last leads to the best results.

Main results. Results in Table 1 indicate that by properly applying the techniques introduced earlier, it is possible to recover the baseline performance with significantly less computation. When architecture remains the same during the second learning phase, better overall performance can be achieved (31.8 vs. 31.6) after 30% of the updates required by the baseline model M25. The gains come from effective learning of both the old and new directions while the latter seems to be better learned than the baseline model (25.2 vs. 24.6). While training into a wider model, applying our techniques recovers the performance with approximately half of the M25$_{wide}$ computation. Although we did not fully recover the performance while training into a deeper model, over 98% of the baseline performance can be achieved using our techniques with half of the baseline M25$_{deep}$ computation. We find that the major degradation in Mt25$_{deep}$ comes from the worse performance on the original directions (33.9 vs. 34.5) which suggests mitigating catastrophic forgetting is harder when expanding the network by depth than by width.

4 Effect of each technique

4.1 Ablation study

To understand the effect of each introduced technique, we conduct an ablation study where each model is trained on the same configuration except for one essential element (i.e., proper initialization, learning rate scaling, data up-sampling). As a naive baseline, while having scaled learning rate and data upsampling, we include the models where no weights from the seed M20 model are used ("random init all") to compare with a less naive baseline "random init new" where the M20 weights are reused and only the newly added parameters (i.e., new token embeddings in all three models, new fully-connected layer weights in Mt25$_{wide}$ and weights of new layers in Mt25$_{deep}$) are randomly initialized. To summarize, each configuration in Table 3 corresponds to the following:

Random init all : All parameters are initialized randomly while model is trained with data up-sampling and learning rate scaling.

Random init new : Newly added parameters are initialized randomly while weights of M20 are copied to the new model. Model is trained with data up-sampling and learning rate scaling.

No up-sampling : Model weights are properly initialized and their learning rates are scaled during training. No language pair is up-sampled.

No lr scaling : Model weights are properly initialized and low-resource pairs are up-sampled whereas no learning rate scaling is applied.

Results in Table 3 confirm the contribution of each of the three introduced techniques in achieving the desired performance across different settings. Overall, not reusing the M20 weights leads to worse performance than the baseline by 2∼6 BLEU in different settings. While reusing the old model's weights, also having proper initialization of the *new* parameters yields better performance than simply initializing with default normal distribution. The benefit is most obvious when training into a wider model (29.9 vs. 32.6) compared to the other two settings. We also observe that data up-sampling is crucial to achieving good performance on the new directions. Not applying up-sampling degrades around ↓ 3BLEU on new directions across all settings, while barely or just slightly hurting the performance on the old directions. On the other hand, not applying learning rate scaling harms the performance of old directions across all settings, which suggests the effectiveness of scaling the learning rate to mitigate catastrophic forgetting, about which we include a more detailed analysis in section 5.2.[8] Since learning rate scaling helps counteract catastrophic forgetting and up-sampling speeds up learning of the new directions, we discover that their effectiveness are additive – better performance can be achieved on both old and new directions by combining these two techniques.

4.2 Effect of proper initialization

In this section we briefly discuss different variations we attempted for initializing new parameters and present the results in Table 4.

Mt25 In the case of having only mismatched vocabularies, we find that dropping the entire old embedding table ("all emb random") causes large drop in performance, and that initializing the new embeddings with that of <unk> leads to slightly better performance than random initialization.

---

[8]The overall differences are small as the results are averaged over 50 directions, however, the paired T-Test shows statistically significant improvements.

[9]We did not apply learning rate scaling in this setup and do not up-sample the old related low-resource languages. All evaluations are done after 30K updates.

	All	**Orig.**	**Added**
Mt25			
all emb random	30.7	32.1	25.1
new random	31.5	33.0	25.5
new <unk>	**31.5**	**33.0**	**25.7**
Mt25$_{wide}$			
new random	31.1	32.3	26.1
linear interp	31.8	33.3	26.0
concat	**32.0**	**33.5**	**26.1**
M20t25$_{deep}$			
random	31.5	32.8	26.3
closest layer	31.6	32.9	26.3
average layer	**31.6**	**32.9**	**26.7**

Table 4: Initializing the new embeddings with that of <unk>, concatenating the original weight with a noisy version of itself in Mt25$_{wide}$ and initializing new layer in encoder/decoder with the averaged encoder/decoder weights in Mt25$_{deep}$ achieve the best performance.[9]

Mt25$_{wide}$ A naive method to initialize wider feed-forward layer projections is to expand the old weight matrix with a randomly initialized weight matrix. However, having random additional parameters messes up the output of feed-forward layer, which interferes with the inter-dependency among layers, thus we also tried concatenating the old weight matrix with a noisy version of itself (concat[10]) followed by a normalization operation[11] to keep the output as close to the original projection output as possible. In addition, rather than maintaining the old weights in block, we also tried linearly interpolating the original weight matrices along the expanded dimension (linear interp). It's important that the new parameters in the two projection matrices in a feed-forward layer match each other along the hidden dimension axis. Results indicate that weight concatenation is a simple and effective way that allows for faster convergence.

Mt25$_{deep}$ Our preliminary experiments show that inserting new layers at the bottom of the encoder and at the top of the decoder is more effective than other attempted scenarios, so we went with this setup in all our Mt25$_{deep}$ experiments. We tried dif-

---

[10]We inject zero-mean Gaussian noise with $std = 0.01$. We also tried not adding noise to the new parameters, which has almost identical performance. To avoid both parts of the weight matrices learning redundant information, we decide to add small perturbation to the new parameters.

[11]We normalize the new weight matrix such that it has similar Frobenius norm as the old weight matrix.

Up-sampling config.	**All**	**Orig.**	**Added**
$\alpha = 1$ (No up-sampling)	31.0	**33.2**	22.2
$\alpha = 5$	31.5*	33.0*	**25.7***
$\alpha = 10$	31.1	32.6*	25.5*
$\alpha = 5$ + rel. low-resource	**31.5***	33.1	25.1*

Table 5: Performance of Mt25 at 30K updates without applying learning rate scaling. Up-sampling new languages with a reasonable ratio during continual learning leads to large gains on these new directions. * indicates p-value of paired T-test against the baseline (top row) is smaller than 0.05.
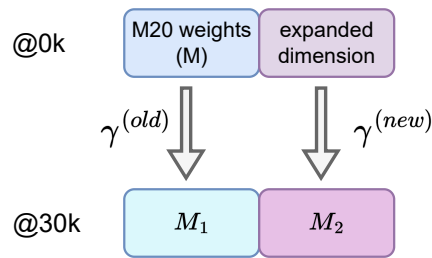

Figure 2: Before the continual learning (@0k), the wider hidden projection matrix is initialized with a concatenation of M20 weights M and a noisy version of itself. During the continual learning stage, the learning rate for weights copied from old model is scaled by $\gamma^{(old)}$, the rest are scaled by $\gamma^{(new)}$.

ferent methods of initializing the inserted layers, including random initialization, copying parameters from the closest layer,[12] and averaging the weights across all encoder or decoder layers. Although the benefit is small, we do perceive that initializing the new layers with averaged layer weights results in better performance, especially on new translation directions (26.7 vs. 26.3).

4.3 Effect of data up-sampling

We multiply the original dataset size of each up-sampled direction by a value α before computing the final up-sample ratio. The new directions thus constitute a larger portion in the new dataset than if not. We show in Table 5 the effect of the up-sample factor α. We find that up-sampling selected languages leads to much better performance (+3BLEU) on new directions than without up-sampling. However, up-sampling the new directions too much (e.g., $\alpha = 10$) worsens the performance on old directions while not improving the new directions. In general, we find the up-sampling

---

[12]Since we include the new layer at the bottom of the encoder and at the top of the decoder, the closest layer means the first encoder layer and last decoder layer respectively.

	All	**Orig.**	**Added**
$\gamma^{(old)} = 1, \quad \gamma^{(new)} = 1$	32.0	33.5	26.1
$\gamma^{(old)} = 0.5, \gamma^{(new)} = 5$	**32.3***	**33.8***	**26.2**
$\gamma^{(old)} = 0.5, \gamma^{(new)} = 0.5$	31.5*	33.3	24.7*
$\gamma^{(old)} = 0.5, \gamma^{(new)} = 1$	32.1*	33.6*	26.0
$\gamma^{(old)} = 5, \quad \gamma^{(new)} = 5$	31.9	33.4	26.1

Table 6: Performance of Mt25$_{wide}$ after 30K updates while applying different learning rate scaling factor based on notation in Figure 2. These experiments adopt an earlier setup where the old related low-resource languages are not up-sampled. * indicates p-value of paired T-test against the baseline (top row) is smaller than 0.05.

factor $\alpha = 5$ adopted in previous studies (Garcia et al., 2021; Berard, 2021) suitable to many other variants we have attempted in this work. Besides merely up-sampling the new directions, we also up-sample the old low-resource directions that belong to the same language family as any of the new directions. Doing so slightly improves the performance on old directions whereas causes a drop in new directions, which further confirms the critical role of up-sampling in balancing the performance between the old and new directions.

4.4 Effect of learning rate scaling

As described in section 2, we scale down the learning rate for old (reused) parameters. In the case of Mt25, all parameters are updated with a smaller learning rate than the new token embeddings. In the wider network, as we have established the effectiveness of using concatenated weights, we apply different learning rate to each part of the weight matrices as generically illustrated in Figure 2. In Mt25$_{deep}$, the new layers are updated with a larger learning rate while the rest of the parameters receive a smaller learning rate. Table 6 suggests that having a smaller learning rate for old parameters is more favorable than scaling all parameters by the same amount ratio. Scaling down the update for all parameters slows down the learning of the new directions, whereas scaling up with a larger value does not improve the performance either. While already applying smaller update on old parameters, scaling up the learning rate for the new parameters can in fact improve both the old and new directions (the top two rows in Table 6). Overall we find that learning rate scaling is an effective and easy-to-implement alternative to previous methods (Kirkpatrick et al., 2017) in terms of alleviating catastrophic forgetting.

	Mt25	**Mt25_v1**	**Mt25_v2**	**M12t25**
All	32.6	32.7	32.5	32.7
High	35.8	36.0	35.8	35.9
Mid	29.6	29.8	29.7	29.3
Low	31.5	31.2	31.4	31.6
V_Low	19.0	20.0	19.0	20.1

Table 7: Besides the language breakdown shown in Table 2, we show our approach also generalizes to other groupings of old and new languages (Mt25_v1, Mt25_v2 and M12t25), details about which are included in Appendix A Table 10.

	Mt25		**Mt25_v1**		**Mt25_v2**	
	ΔBLEU	Up	ΔBLEU	Up	ΔBLEU	Up
eng-bel	-1.2	N	2.0	Y	2.4	Y
eng-guj	1.6	Y	0.9	Y	0.0	N
eng-npi	0.8	Y	-0.1	Y	-2.3	N
eng-xho	1.3	Y	0.0	N	1.1	Y
bel-eng	-1.1	N	-0.5	Y	-0.7	Y
guj-eng	-2.5	Y	-2.6	Y	-2.2	N
npi-eng	-1.6	Y	-0.6	Y	-1.9	N
xho-eng	-1.1	Y	-1.3	N	-0.7	Y

Table 8: Zoomed in analysis over specific language pairs. Our approach is more effective in learning eng→xxx directions than xxx→eng directions across different seed-language setups regardless if the language is up-sampled (Up=Y) or not (Up=N).

5 Analysis

5.1 How does the choice of seed languages affect continual learning?

In addition to the core setup described in Table 2, we also experiment with other settings. We considered three new setups following the natural scenario where the seed languages are mostly of high- and mid-resource languages. Two of the new setups still adopt the 20/5 division between old and new languages, but emphasize different scenarios — Mt25_v1 covers one mid resource language in the set of new languages and Mt25_v2 does not add any new script. The last one instead initializes from a model trained on 12 high- and mid-resource languages and adds in 13 new languages.[13] We follow the same configuration used in the previous experiments and train into wider models in the continual learning phase. Results in Table 7 demonstrate that our approach successfully generalizes to all the new settings, as similar performances can be achieved

---

[13]For detailed breakdown of these three new settings, we refer readers to Table 10 and Table 11 in Appendix A
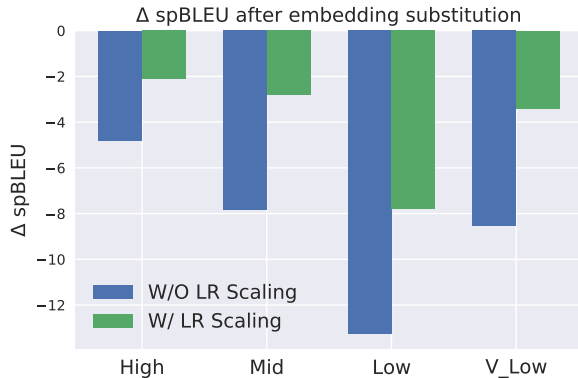
Figure 3: We measure the drop in spBLEU on original directions after substituting overlapped embeddings in $Mt25_{wide}$ back to M20 model. Applying learning rate scaling helps alleviate catastrophic forgetting as spBLEU drops less than without LR scaling.
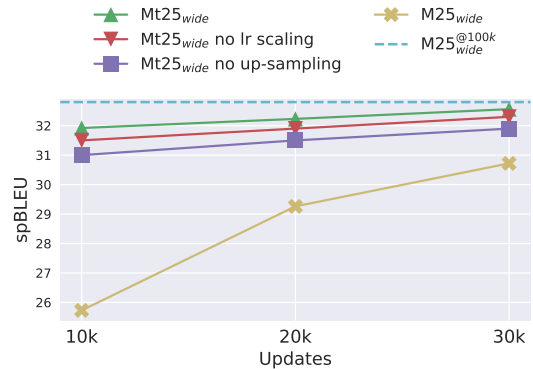

Figure 4: Training the M25 from scratch without using the M20 weights converges significantly slower. Applying the techniques introduced in this work results in the largest computation reduction, achieving over 95% performance with less than 10% baseline computation.

when models are trained on different choices of seed languages, even when the number of seed languages are different. However, we do observe small differences after zooming in by resource-levels. A more fine-grained analysis over a few low-resource languages that cover diverse scripts (bel-Cyrl, guj-Gujr, npi-Deva, xho-Latn) is presented in Table 8. In general, regardless of if the language pair is in the seed language set or not, the eng→xxx directions are learned faster than xxx→eng directions, since most of the pairs already exceed the baseline performance after just 30K updates (compare the upper part of Table 8 to the lower part). We also verify the effectiveness of data up-sampling for eng→xxx directions. Up-sampling these pairs leads to much better performance than when not up-sampling (e.g., compare performance of eng-bel in the three setups).

5.2 How effective is learning rate scaling in mitigating catastrophic forgetting?

To quantitatively measure the amount of information lost after the continual learning phase, we adopt an evaluation setup akin to (Garcia et al., 2021), in which the embeddings in M20 that overlap with Mt25 are substituted with the corresponding embeddings in Mt25. One can evaluate this new M20 model with substituted embeddings on the original 20 languages, and use the drop in spBLEU as a proxy for the amount of knowledge lost in the embeddings due to catastrophic forgetting. In Figure 3, we display the spBLEU drop after substituting the embeddings of a $Mt25_{wide}$ model trained with and without learning rate scaling. For both

variants, the spBLEU scores drop, indicating that some information in the embeddings is lost after training on the new languages. We also find that, for both models, the decrease in spBLEU is larger for (very) low-resource languages than high and mid resource languages, which suggests (very) low-resource languages are easier to be "forgotten" in the second training phase, which reinforces our decision of also up-sampling the related low-resource languages as introduced in section 4.3. Finally, not applying learning rate scaling leads to much larger decrease in all directions, which manifests the effectiveness of scaling down the learning rate for alleviating catastrophic forgetting.[14]

5.3 Computation saved

Due to the mismatched vocabularies and architectures, models trained on the combination of new and old languages are typically re-trained from scratch after already incurring large computation on old languages. In this section, we look at how much computation can be effectively saved with the approach proposed in this paper. Note that, for both our methods and the retraining approach, M20 is already trained, and its computation cost is excluded from our calculation.

Although results in Table 1 shows that 50% of the baseline computation is required to recover the baseline performance, we show in Figure 4 that much computation can be saved if we slightly slack the target performance. The $M25_{wide}$ model

---

[14]Besides probing the effect of learning rate scaling on embeddings, we also present an analysis on the expanded feed-forward weights in Appendix C.

trained from scratch reaches only 78% of the baseline performance after 10k updates, while models trained with the combination of our techniques achieve over 97% of the baseline performance after the same amount of updates (10% total computation of training Mt25$_{wide}$).

6   Related Work

Our work is closely related to prior research on adapting existing MMT models (Mohammadshahi et al., 2022) to new languages (Lakew et al., 2019; Kocmi, 2020). Neubig and Hu (2018) add low-resource languages to multilingual models by fine-tuning on low-resource data while regularizing with related high-resource data. Garcia et al. (2021) introduce simple vocabulary substitution for adapting MMT models to new languages without any architectural changes. Another line of research employs modular approaches, which include training lightweight adapters (Bapna and Firat, 2019), language-specific encoder-decoders (Escolano et al., 2019, 2021), language specific embeddings (Berard, 2021) for learning new languages. While sometimes escaping the need to train on old examples, growing the model in a modular fashion (Rusu et al., 2016) requires non-trivial changes to standard architectures. In contrast, our work relies on rehearsal mechanism (i.e., also train on old examples) but does not need to modify network structures (Robins, 1995).

Our approach is also related to works that focus on continual learning of MT models for adapting multiple domains. Thompson et al. (2019); Gu and Feng (2020) adopt a method derived from Elastic Weight Consolidation (Kirkpatrick et al., 2017) to alleviate catastrophic forgetting. While most prior works only investigate two-stage continual learning, Cao et al. (2021) propose a new framework that extends to multi-stage training to mitigate catastrophic forgetting (Ring, 1994). The initialization of the new parameters and embeddings in our technique is also related to that in (Pfeiffer et al., 2021), which accommodate multilingual models to unseen scripts via matrix factorization. Our focus on the architectural differences between initial and continual learning phase is also relevant to recent discoveries that wider networks forget less catastrophically (Mirzadeh et al., 2022).

7   Conclusion

We show in this work that it is possible to efficiently bootstrap from existing models and recover the baseline performance with much less computation while vocabularies and architectures can be different in the continue learning stage. We highlight the importance of (1) reusing the existing model weights and carefully initializing the new parameters, (2) applying learning rate scaling, and (3) performing data up-sampling. Analyses reveal that scaling down the learning rate for old parameters helps alleviate catastrophic forgetting, and that data up-sampling is vital to achieving good performance on the new directions. We hope our work can help save computation for research into large-scale multilingual MT models, and more generally, will help spur research into continual multitask learning in the presence of architectural changes.

Limitations

While we explore the under-studied architectural mismatches for continual learning of MMT models, we focus exclusively on adding new languages in bulk, without investigating adding languages one by one continuously. Furthermore, due to limited computational resources, we only experimented with a few typical scenarios where the new languages are low or very-low resourced. Experiments on other groupings of old and new languages could further validate the effectiveness of our approach.

Acknowledgement

References

Roee Aharoni, Melvin Johnson, and Orhan Firat. 2019. Massively multilingual neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3874–3884, Minneapolis, Minnesota. 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.

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.

Alexandre Berard. 2021. Continual learning in multilingual NMT via language-specific embeddings. In *Proceedings of the Sixth Conference on Machine Translation*, pages 542–565, Online. Association for Computational Linguistics.

Yue Cao, Hao-Ran Wei, Boxing Chen, and Xiaojun Wan. 2021. Continual learning for neural machine translation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3964–3974, Online. Association for Computational Linguistics.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. 2015. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1723–1732.

Carlos Escolano, Marta R. Costa-jussà, and José A. R. Fonollosa. 2019. From bilingual to multilingual neural machine translation by incremental training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 236–242, Florence, Italy. Association for Computational Linguistics.

Carlos Escolano, Marta R. Costa-jussà, José A. R. Fonollosa, and Mikel Artetxe. 2021. Multilingual machine translation: Closing the gap between shared and language-specific encoder-decoders. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 944–948, Online. Association for Computational Linguistics.

Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, Naman Goyal, Tom Birch, Vitaliy Liptchinsky, Sergey Edunov, Edouard Grave, Michael Auli, and Armand Joulin. 2020. Beyond english-centric multilingual machine translation.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-way, multilingual neural machine translation with a shared attention mechanism. *arXiv preprint arXiv:1601.01073*.

Robert French. 1993. Catastrophic interference in connectionist networks: Can it be predicted, can it be prevented? In *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann.

Xavier Garcia, Noah Constant, Ankur Parikh, and Orhan Firat. 2021. Towards continual learning for multilingual machine translation via vocabulary substitution. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1184–1192, Online. Association for Computational Linguistics.

Naman Goyal, Cynthia Gao, Vishrav Chaudhary, Peng-Jen Chen, Guillaume Wenzek, Da Ju, Sanjana Krishnan, Marc'Aurelio Ranzato, Francisco Guzmán, 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.

Shuhao Gu and Yang Feng. 2020. Investigating catastrophic forgetting during continual training for neural machine translation. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4315–4326, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Francisco Guzmán, Peng-Jen Chen, Myle Ott, Juan Pino, Guillaume Lample, Philipp Koehn, Vishrav Chaudhary, and Marc'Aurelio Ranzato. 2019. Two new evaluation datasets for low-resource machine translation: Nepali-english and sinhala-english. In *arXiv preprint arXiv:1902.01382*.

Thanh-Le Ha, Jan Niehues, and Alexander Waibel. 2016. Toward multilingual neural machine translation with universal encoder and decoder. *arXiv preprint arXiv:1611.04798*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Tom Kocmi. 2020. Exploring benefits of transfer learning in neural machine translation. *ArXiv*, abs/2001.01622.

Tom Kocmi, Christian Federmann, Roman Grundkiewicz, Marcin Junczys-Dowmunt, Hitokazu Matsushita, and Arul Menezes. 2021. To ship or not to ship: An extensive evaluation of automatic metrics for machine translation. In *Proceedings of the Sixth Conference on Machine Translation*, pages 478–494, Online. Association for Computational Linguistics.

Taku Kudo. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Surafel M. Lakew, Alina Karakanta, Marcello Federico, Matteo Negri, and Marco Turchi. 2019. Adapting multilingual neural machine translation to unseen languages. In *Proceedings of the 16th International Conference on Spoken Language Translation*, Hong Kong. Association for Computational Linguistics.

Surafel Melaku Lakew, Aliia Erofeeva, Matteo Negri, Marcello Federico, and Marco Turchi. 2018. Transfer learning in multilingual neural machine translation with dynamic vocabulary. In *IWSLT*.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Seyed Iman Mirzadeh, Arslan Chaudhry, Dong Yin, Timothy Nguyen, Razvan Pascanu, Dilan Gorur, and Mehrdad Farajtabar. 2022. Architecture matters in continual learning. *arXiv preprint arXiv:2202.00275*.

Alireza Mohammadshahi, Vassilina Nikoulina, Alexandre Berard, Caroline Brun, James Henderson, and Laurent Besacier. 2022. SMaLL-100: Introducing shallow multilingual machine translation model for low-resource languages. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8348–8359, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Graham Neubig and Junjie Hu. 2018. Rapid adaptation of neural machine translation to new languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 875–880, Brussels, Belgium. Association for Computational Linguistics.

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.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2021. UNKs everywhere: Adapting multilingual language models to new scripts. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10186–10203, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Maja Popović. 2017. chrF++: words helping character n-grams. In *Proceedings of the Second Conference on Machine Translation*, pages 612–618, Copenhagen, Denmark. Association for Computational Linguistics.

Mark Bishop Ring. 1994. *Continual Learning in Reinforcement Environments*. Ph.D. thesis, Graduate School of The University of Texas at Austin, USA. UMI Order No. GAX95-06083.

Anthony Robins. 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

Brian Thompson, Jeremy Gwinnup, Huda Khayrallah, Kevin Duh, and Philipp Koehn. 2019. Overcoming catastrophic forgetting during domain adaptation of neural machine translation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2062–2068, Minneapolis, Minnesota. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
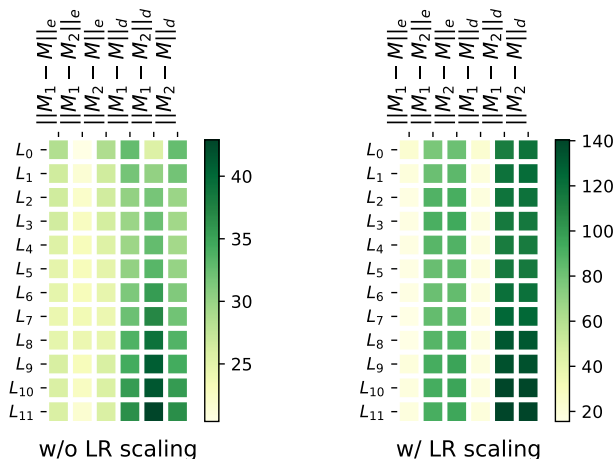
Figure 5: We measure the amount of change in each feed-forward layer's projection weights via Frobenius norm ($\|\cdot\|_e$ for encoder layers and $\|\cdot\|_d$ for decoder layers). Notation in this figure follows that in Figure 2. Scaling down the learning rate for parameters copied from the old model keeps them close to their initialization while pushes the newly added parameters farther away.

A Language details

We present two extra language grouping settings we have experimented in this section. To verify the validity of our approach on other language groupings, we also experiment with two other settings as shown in Table 10. In addition to dividing the old/new languages by 20/5, we also tried a setting where the seed model is trained on fewer languages as reported in Table 11.

B Detailed performance of each direction

Table 12 contains the detailed performance of all translation directions for all models reported in Table 1.

C Norm Analysis

The analysis in section 5.2 is limited to only measuring the lost information in the embeddings, to understand how learning rate scaling affects the weights other than embeddings, one natural extension is substituting other weights back to M20 as well. However, it leads to much worse performance for both variants as the inter-dependency among layers is impaired in this case. Therefore, we instead measure the amount that the weights of each encoder or decoder layer (denoted with $L_0, L_1, \ldots, L_{11}$) change in the latent space via the Frobenius norm. Following the notation in Figure 2, we measure how much the weight matrices M_1 and

M_2 have changed from the original weight matrix M ($\|M_1 - M\|$ and $\|M_2 - M\|$), as well as how much they differ from each other ($\|M_1 - M_2\|$). We refer to the Frobenius norm in an encoder or a decoder layer with $\|\cdot\|_e$ and $\|\cdot\|_d$ respectively. The trend in Figure 5 shows that applying learning rate scaling prevents M_1 from deviating too much from the original weights M and at the same time pushes the new parameters to space farther from its initialization. This is in contrast with the smaller differences when not applying learning rate scaling. The left side of Figure 5 indicates that even after the continual learning, M_1 and M_2 stay close to each other across the encoder layers, it is only when reaching the last several decoder layers do the two matrices demonstrate larger differences. Since we initialize M_1 and M_2 both based on M, having larger $\|M_1 - M_2\|$ reduces the chance of both parts learning redundant information i.e., effectively using the additional parameters.

D Scaling learning rate by Fisher information

Besides multiplying the learning rate for all old parameters with the same scaling factor, we also tried scaling the learning rate based on their Fisher Information. This is directly inspired by Elastic Weight Consolidation (Kirkpatrick et al., 2017), in which extra penalty is incurred when parameters crucial to the old tasks deviate too much from their original values. We plot the distribution of the per-token Fisher information of each parameter in Figure 6 (right). We further experiment with LR scaling over selected parameters that are supposed to be important for old tasks (Fisher information exceeds certain threshold based on Figure 6 (right)). Results in Table 11 (right) shows that among our attempted settings, scaling only part of the parameters based on Fisher information does not improve the overall performance. We conjecture that the performance could be improved if the Fisher Information is calculated on a larger set and that if we apply a piece-wise threshold function for scaling the learning rate of different parameters.

E Effect of LR scaling in alleviating catastrophic forgetting

We include the spBLEU drop on the old languages after substituting the embeddings of Mt25 back to M20 in Figure 6.

	Language	Language code	Language family	# bitext	Res. level	Script
M20 old languages	Latvian	lav	Baltic	2,288,307	Mid	Latn
	Lithuanian	lit	Baltic	2,322,619	Mid	Latn
	Swahili	swh	Benue-Congo	1,133,371	Mid	Latn
	Zulu	zul	Benue-Congo	656,450	Low	Latn
	Ukrainian	ukr	East Slavic	925,922	Low	Cyrl
	Belarusian	bel	East Slavic	55,185	V_Low	Cyrl
	Russian	rus	East Slavic	13,962,768	High	Cyrl
	Hindi	hin	Indo-Iranian	1,032,502	Mid	Deva
	Marathi	mar	Indo-Iranian	181,642	Low	Deva
	Malay	msa	Malayo-Polynesian	305,469	Low	Latn
	Macedonian	mkd	South Slavic	403,742	Low	Cyrl
	Bulgarian	bul	South Slavic	2,118,839	Mid	Cyrl
	Kyrgyz	kir	Turkic	336,554	Low	Cyrl
	Finnish	fin	Uralic	3,764,770	Mid	Latn
	Estonian	est	Uralic	2,410,543	Mid	Latn
	Polish	pol	West Slavic	3,062,818	Mid	Latn
	Czech	ces	West Slavic	23,792,604	High	Latn
	French	fra	Western European	20,031,051	High	Latn
	Spanish	spa	Western European	16,606,594	High	Latn
	German	deu	Western European	10,198,897	High	Latn
Mt25 new languages	Xhosa	xho	Benue-Congo	634,078	Low	Latn
	Gujarati	guj	Indo-Iranian	153,985	Low	Gujr
	Nepali	npi	Indo-Iranian	72,250	V_Low	Deva
	Indonesian	ind	Malayo-Polynesian	771,801	Low	Latn
	Kazakh	kaz	Turkic	627,734	Low	Cyrl

Table 9: Detailed information about languages used in the main setup of our experiments. Languages having >10M examples are of high-resource languages, having (1M, 10M] mid-resource languages. The rest are of low-resource languages, v_low is a subset of low-resource languages that have <100K examples.

Mt25_v1 Original				Added	Mt25_v2 Original				Added
lav	ukr*	msa	pol	guj	lav	hin	mkd	pol	zul
lit	rus	mkd	ces	npi	lit	mar	bul	ces	xho
swh	hin	bul	fra	est	swh	guj	kaz*	fra	ind
zul	mar*	fin	spa	bel	ukr*	npi	fin	spa	bel
xho	ind	kaz*	deu	kir	rus	msa*	est	deu	kir

Table 10: Besides the breakdown between old and new languages as shown in Table 2, Mt25_v1 uses the left grouping and Mt25_v2 uses the right grouping of languages. * marks the languages that are also up-sampled during continual learning.

Original		Added	
lav	est	zul	mar
lit	pol	xho	guj
swh	ces	ukr	npi
rus	fra	bel	ind
bul	spa	hin	msa
fin	deu	mkd	kaz
		kir	

	All	Orig.	Added
f=1e-6, $\gamma^{(old)}$=0.25	32.3	33.8	26.4
f=1e-6, $\gamma^{(old)}$=0.1	32.1	33.7	26.0
f=1e-5, $\gamma^{(old)}$=0.25	31.9	33.3	26.2
f=1e-5, $\gamma^{(old)}$=0.1	32.1	33.5	26.6
f=1e-5, $\gamma^{(old)}$=0.05	31.8	33.3	25.8
f=1e-4, $\gamma^{(old)}$=0.25	31.7	33.1	26.1

Table 11: **Left:** M12t25 starts from a model trained on 12 high- and mid- resource languages and grows to a wider model to support 13 new low- and very-low- resource languages. None of old languages is up-sampled in the continual learning stage. **Right:** Performance of Mt25$_{wide}$ after 30k updates when the learning rate for parameter whose Fisher information is greater than threshold f is scaled down by the corresponding $\gamma^{(old)}$ in each row.
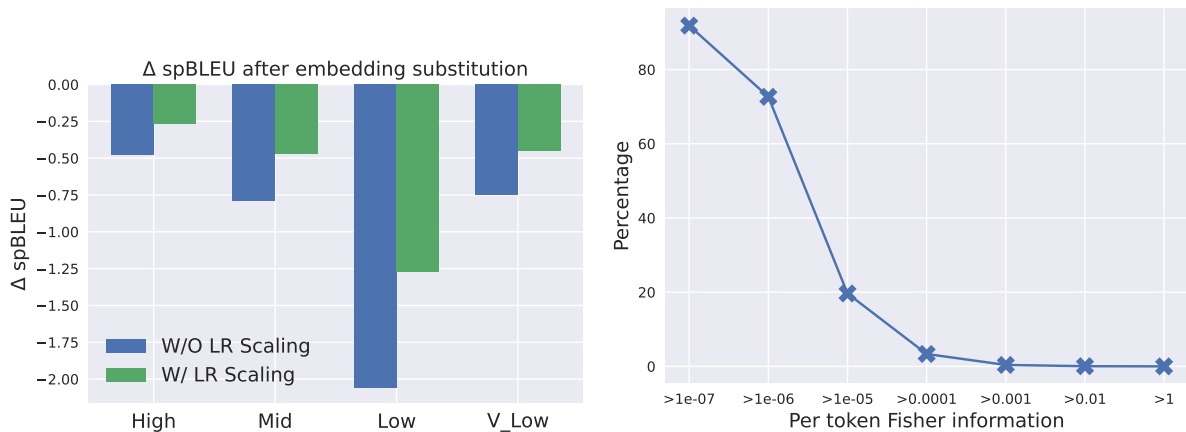
Figure 6: **Left:** spBLEU drop after substituting the embedding of the learned Mt25 model back to M20 model. **Right:** Per token Fisher information calculated over FloRes dev set.

Lang_pair	M25		Mt25		$M25_{wide}$		$Mt25^{(@30k)}_{wide}$		$M25_{deep}$		$Mt25^{(@30k)}_{deep}$	
	spBLEU	chrF++	spBLEU	chrF++	spBLEU	chrF++	spBLEU	chrF++	spBLEU	chrF++	spBLEU	chrF++
bel-eng	18.1	0.74	18.3	0.74	20.1	0.74	19.0	0.74	19.9	0.74	18.5	0.74
bul-eng	40.8	0.81	41.1	0.81	41.7	0.81	41.5	0.81	41.9	0.81	41.0	0.81
ces-eng	42.6	0.81	42.3	0.81	42.9	0.82	42.5	0.81	43.0	0.81	42.0	0.81
deu-eng	44.7	0.82	44.5	0.82	45.3	0.82	45.2	0.82	45.4	0.82	44.5	0.82
eng-bel	11.7	0.59	11.3	0.59	13.2	0.61	12.1	0.59	13.1	0.60	12.1	0.60
eng-bul	39.5	0.78	40.3	0.79	40.5	0.79	40.1	0.78	40.6	0.79	40.3	0.79
eng-ces	38.4	0.76	38.5	0.76	38.4	0.76	38.2	0.76	38.8	0.76	38.4	0.76
eng-deu	39.0	0.79	39.3	0.79	40.1	0.79	39.2	0.78	39.8	0.79	39.3	0.79
eng-est	30.3	0.76	30.1	0.76	32.0	0.77	30.7	0.76	31.5	0.76	30.7	0.76
eng-fin	28.5	0.76	28.5	0.76	30.3	0.76	29.3	0.76	30.7	0.77	29.5	0.76
eng-fra	49.4	0.81	49.4	0.81	48.6	0.81	49.3	0.81	49.1	0.81	49.1	0.81
eng-guj	24.2	0.63	28.3	0.67	25.5	0.64	28.4	0.67	25.3	0.64	28.0	0.66
eng-hin	32.0	0.71	31.5	0.71	33.4	0.72	32.6	0.71	32.7	0.71	32.8	0.71
eng-ind	40.1	0.81	40.0	0.81	41.2	0.81	44.4	0.82	41.4	0.81	43.7	0.82
eng-kaz	14.6	0.58	13.4	0.57	14.8	0.58	18.0	0.61	16.4	0.60	15.6	0.59
eng-kir	18.0	0.61	16.4	0.59	19.1	0.62	20.9	0.63	18.5	0.62	20.4	0.63
eng-lav	32.4	0.75	33.5	0.75	33.6	0.75	34.0	0.75	34.1	0.75	33.6	0.75
eng-lit	28.3	0.73	28.1	0.73	29.5	0.74	28.9	0.74	29.7	0.74	28.6	0.73
eng-mar	16.9	0.62	16.3	0.62	18.2	0.62	17.8	0.63	18.5	0.63	17.3	0.62
eng-mkd	34.9	0.77	35.2	0.77	36.5	0.77	35.8	0.77	35.8	0.77	35.2	0.77
eng-msa	37.2	0.80	32.5	0.79	39.6	0.81	43.3	0.82	39.3	0.80	42.3	0.82
eng-npi	11.3	0.47	17.6	0.59	13.9	0.52	16.4	0.57	13.7	0.52	15.3	0.54
eng-pol	23.5	0.70	23.5	0.70	24.9	0.70	24.2	0.70	24.4	0.70	23.3	0.70
eng-rus	32.4	0.71	32.3	0.70	33.1	0.71	33.2	0.70	33.5	0.71	32.5	0.70
eng-spa	29.8	0.74	28.9	0.74	29.2	0.74	29.6	0.74	30.2	0.74	29.5	0.74
eng-swh	34.4	0.78	34.5	0.78	35.3	0.78	35.2	0.78	35.1	0.78	35.0	0.79
eng-ukr	31.4	0.72	31.7	0.72	33.2	0.72	32.1	0.72	33.5	0.72	32.1	0.72
eng-xho	21.9	0.73	21.7	0.73	22.3	0.73	24.2	0.75	22.9	0.73	23.4	0.74
eng-zul	30.4	0.76	29.1	0.75	31.3	0.76	33.8	0.77	31.3	0.76	33.1	0.77
est-eng	36.8	0.78	36.7	0.78	37.6	0.79	36.3	0.78	37.4	0.79	36.9	0.79
fin-eng	34.5	0.78	35.1	0.78	35.6	0.78	35.1	0.78	35.8	0.78	35.0	0.78
fra-eng	47.0	0.83	47.2	0.83	48.2	0.83	47.9	0.83	48.2	0.83	47.5	0.83
guj-eng	29.9	0.76	29.7	0.76	31.5	0.76	30.7	0.76	31.4	0.76	29.0	0.75
hin-eng	34.9	0.79	35.0	0.79	36.8	0.80	35.7	0.79	36.6	0.79	35.4	0.79
ind-eng	42.2	0.81	41.4	0.81	43.8	0.81	42.8	0.81	43.4	0.81	43.0	0.81
kaz-eng	1.5	0.46	2.8	0.48	1.3	0.47	4.8	0.50	3.1	0.47	3.4	0.47
kir-eng	19.9	0.69	19.4	0.69	20.9	0.70	20.3	0.69	20.7	0.70	19.9	0.69
lav-eng	35.4	0.79	35.4	0.78	36.6	0.79	36.2	0.78	36.7	0.79	35.4	0.79
lit-eng	33.3	0.77	33.0	0.77	34.4	0.77	33.2	0.76	34.0	0.77	33.0	0.77
mar-eng	28.2	0.75	27.9	0.75	30.3	0.76	28.6	0.75	30.2	0.76	28.2	0.75
mkd-eng	41.9	0.81	42.6	0.81	43.8	0.82	42.8	0.81	43.2	0.81	42.9	0.81
msa-eng	43.4	0.81	42.3	0.81	44.2	0.82	44.6	0.82	43.9	0.81	43.8	0.82
npi-eng	29.2	0.75	29.6	0.75	31.7	0.76	29.7	0.75	31.7	0.76	29.6	0.75
pol-eng	31.5	0.77	31.6	0.76	32.4	0.77	31.9	0.76	32.1	0.77	31.6	0.76
rus-eng	36.9	0.79	36.6	0.79	38.1	0.79	37.6	0.79	38.0	0.79	36.9	0.79
spa-eng	33.1	0.79	33.4	0.79	33.7	0.79	33.7	0.79	34.1	0.79	33.5	0.79
swh-eng	40.2	0.79	40.7	0.79	42.0	0.79	40.8	0.79	41.9	0.79	40.6	0.79
ukr-eng	38.6	0.80	39.0	0.80	40.5	0.80	39.6	0.80	40.4	0.80	39.4	0.80
xho-eng	31.4	0.73	31.2	0.73	33.1	0.74	32.4	0.73	32.8	0.74	31.3	0.73
zul-eng	34.8	0.76	34.4	0.76	36.0	0.76	37.1	0.77	36.2	0.76	36.2	0.76

Table 12: Detailed performance of each translation direction for all models shown in Table 1