Mask More and Mask Later: Efficient Pre-training of Masked Language Models by Disentangling the [MASK] Token

Baohao Liao[♣], **David Thulke**[♦]
Sanjika Hewavitharana[♠], **Hermann Ney**[♦], **Christof Monz**[♣]
[♣]University of Amsterdam, [♦]RWTH Aachen University, [♠]eBay
b.liao@uva.nl, thulke@hltpr.rwth-aachen.de
shewavitharana@ebay.com, ney@informatik.rwth-aachen.de, c.monz@uva.nl

Abstract

The pre-training of masked language models (MLMs) consumes massive computation to achieve good results on downstream NLP tasks, resulting in a large carbon footprint. In the vanilla MLM, the virtual tokens, [MASK]s, act as placeholders and gather the contextualized information from unmasked tokens to restore the corrupted information. It raises the question of whether we can append [MASK]s at a later layer, to reduce the sequence length for earlier layers and make the pre-training more efficient. We show: (1) [MASK]s can indeed be appended at a later layer, being disentangled from the word embedding; (2) The gathering of contextualized information from unmasked tokens can be conducted with a few layers. By further increasing the masking rate from 15% to 50%, we can pre-train RoBERTa-base and RoBERTa-large from scratch with only 78% and 68% of the original computational budget without any degradation on the GLUE benchmark. When pre-training with the original budget, our method outperforms RoBERTa for 6 out of 8 GLUE tasks, on average by 0.4%. [1]

1 Introduction

Large-scale pre-trained MLMs, like BERT (Devlin et al., 2019) and its variants (Liu et al., 2019; Lan et al., 2020; Clark et al., 2020; Song et al., 2019; Lewis et al., 2020), have achieved great success in various NLP tasks, such as machine translation (Liu et al., 2020; Zhu et al., 2020), general language understanding (Wang et al., 2019b,a), question answering (Rajpurkar et al., 2016), summarization (Liu, 2019) and claim verification (Soleimani et al., 2020). To make the pre-trained model generalize well to a wide range of tasks, MLMs tend to have a large number of parameters, even in the billion scale (Shoeybi et al., 2019), and are trained with plenty of data. This is prohibitively expensive and generates significant amounts of CO_2 emissions (Strubell et al., 2019; Patterson et al., 2021). How to make the pre-training of MLMs more efficient while retaining their superior performance is a critical research question.

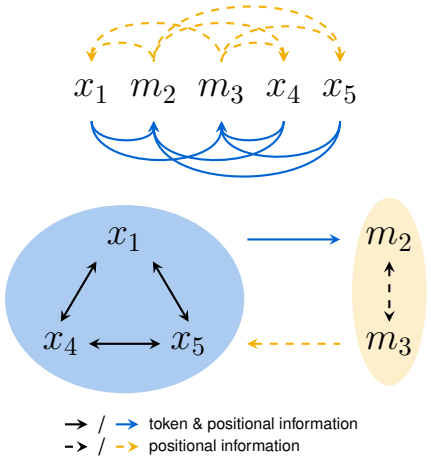

Figure 1: **Information flows of vanilla MLM**. A sentence, $\{x_1, x_2, x_3, x_4, x_5\}$, is corrupted by replacing x_2 and x_3 with a virtual token m indexed with the corresponding positions. [2]

Various attempts on efficient pre-training have obtained effective results. Hou et al. (2022) and Wu et al. (2021) applied a prior knowledge extracted from the MLM itself to make the prediction more inclined to rare tokens. Shoeybi et al. (2019) and You et al. (2020) made use of mixed-precision and distributed training to speed up the pre-training. Data-efficient pre-training objectives (Clark et al., 2020; Lan et al., 2020) and progressively stacking technique (Gong et al., 2019) also work quite well. Orthogonal to these directions, we dive deeply into the information flows of the vanilla MLM, trying to split different types of information flows into multiple stages and making the model spend more computation on the complex flow.

The information transferred among tokens can

[1]Code at https://github.com/BaohaoLiao/3ml

[2]Except where explicitly mentioned, we ignore randomly replaced and unchanged tokens for simplicity.

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 1478–1492
December 7-11, 2022

be split into: *position information* and non-positional information (termed as *token information* in this paper). As shown in Figure 1, the information flows for a corrupted sentence during training consist of: (1) Position and token information among unmasked tokens; (2) Position and token information from unmasked tokens to [MASK]s; (3) Position information among [MASK]s; (4) Position information from [MASK]s to unmasked tokens. These information flows happen in each Transformer block (Vaswani et al., 2017), more specifically in the self-attention module. In addition, the 4^{th} flow brings no additional information given the 1^{st} one, since the positions information from [MASK]s can be inferred implicitly given the positions of the unmasked tokens. We ignore the 4^{th} flow for the following discussion.

Intuitively, the amount of information transferred in each flow is not at the same level. The 3^{rd} flow contains the least information. We make an assumption about the first two flows.

The Information Flow Assumption. *Position and token information among unmasked tokens (1^{st} flow) are more difficult to learn than the transfer of this knowledge to [MASK]s (2^{nd} flow).*

This assumption is empirically proven later (§4.2). Since the difficulty of information transfer varies among different flows, it makes sense to divide the flows into multiple stages, forcing the model to spend more computation on the more complex one.

We propose a two-stage learning method. For the early layers of an MLM, we detach [MASK]s and only input the embedding of unmasked tokens. So the model firstly focuses on the most complex (1^{st}) information flow. At an intermediate layer, we append the embedding of [MASK]s with their corresponding position information back to the sequence. Then the remaining layers of the MLM fuse all information. In this way, the sequence length for the early layers becomes shorter due to excluding [MASK]s. We further reduce the sequence length by increasing the masking rate for higher efficiency. We call our method *mask more and mask later (3ML)*, since [MASK]s are appended later and we have a larger masking rate.

In this work, we introduce two models designed for 3ML (§2), empirically show two prerequisites of efficiency for 3ML hold (§4), conduct extensive experiments to select an optimal setting for high performance and efficiency (§5), and finally compare 3ML's results to strong baselines (§6).

Our main contributions are summarized as: (1) We introduce a simple, intuitive but effective method for the efficient pre-training of MLM; (2) We prove two prerequisites that are important for 3ML; (3) On the GLUE benchmark, 3ML achieves the same performance as RoBERTa-base and RoBERTa-large (Liu et al., 2019) with only 78% and 68% of the original computation budget, and outperform them with the same budget.

2 Model

In this section, we first discuss the information flows in a vanilla MLM, i.e. BERT, then introduce two architectures designed for 3ML (Figure 2).

2.1 Vanilla Masked Language Model

Masked language models reconstruct a sequence with corrupted information. Given a sequence of tokens $\boldsymbol{x} = \{x_t\}_{t=1}^{T}$ with t denoting the token's position, the corrupted version $\hat{\boldsymbol{x}}$ is generated by randomly setting a portion of $\boldsymbol{x}$ to a special symbol [MASK]. MLM is trained to learn the distribution $p(\boldsymbol{x}|\hat{\boldsymbol{x}})$ with a loss function:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[-\sum_{t=1}^{T}\delta_{x_t\neq\hat{x}_t}\log p_\theta(x_t|\hat{\boldsymbol{x}})] \quad (1)$$

where $\delta_{x_t\neq\hat{x}_t}$ is a Kronecker Delta function:

$$\delta_{x_t\neq\hat{x}_t} = \begin{cases} 1 & x_t \neq \hat{x}_t \\ 0 & x_t = \hat{x}_t \end{cases}$$

As shown in Figure 1, position and token information are transferred among different tokens in the model. We can cluster the flows into four types: (1) From unmasked tokens to unmasked tokens (within the blue area): MLM transfers position and token information among unmasked tokens, generating uncorrupted contextualized information; (2) From unmasked tokens to [MASK]s (from the blue area to the yellow area): MLM transfers the uncorrupted contextualized information to [MASK]s; (3) From [MASK]s to [MASK]s (within the yellow area): MLM transfers the position information among [MASK]s. Since all masked tokens have the same token embedding, there is no transfer of token information; (4) From [MASK]s to unmasked tokens (from the yellow area to the blue area): MLM transfers position information from [MASK]s to unmasked tokens.

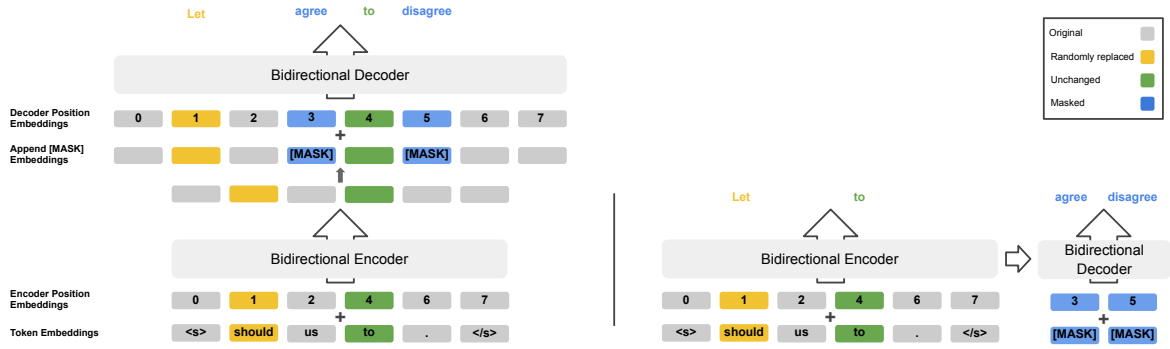

Figure 2: **Overview of 3ML architectures**. A sentence,"<s> Let us agree to disagree. </s>", is corrupted to "<s> should us [MASK] to [MASK]. </s>". Left: 3ML with only self-attention layers ($3ML_{self}$). Right: 3ML with a decoder consisting of both self- and cross-attention layers ($3ML_{cross}$). We achieve efficient pre-training by discarding [MASK]s for the encoder and applying a small decoder for the whole sequence length. For fine-tuning on downstream tasks, the decoder is removed.

2.2 Mask More and Mask Later

Since different flows of the vanilla MLM contain different amounts of information (§1), we propose a two-stage training method where more computation is allocated to the most complex flow, the one among unmasked tokens. At a later stage, we fuse all flows together as vanilla MLM. In this way, we aim to improve the efficiency by reducing the sequence length for the first stage by discarding [MASK]s. Even though at the later stage we still need to fuse all information flows together, back to the original sequence length, we only need to apply a few layers for that, since the 1^{st} flow is already learned quite well during the first stage. Combining a large masking rate and a small number of layers for the second stage together, we can achieve an efficient pre-training. In short, two prerequisites contribute to our efficient pre-training: we can mask more and mask later (test in §4).

We design two architectures, *3ML$_{self}$* and *3ML$_{cross}$*, that only differ from each other on the decoder. As shown in Figure 2, we first input the unmasked tokens to both models. At an intermediate layer, we input the token embedding of [MASK] and fuse all information flows together. With our method, the token embedding of [MASK] is disentangled from the original word embedding space and located in a latent space.

3ML$_{self}$ This architecture is inspired by a computer vision method (He et al., 2021) designed for two-stage learning. $3ML_{self}$ has an encoder and a decoder, with a self-attention Transformer block as the base layer for both. A prediction layer for masked tokens is located at the end of the decoder. Only unmasked tokens are fed into the encoder. So the encoder only transfers information among unmasked tokens (1^{st} flow). After the encoder, we append the token embedding of [MASK] back to the sequence with a new positional embedding and input to the decoder. Since the input sequence to the decoder consists of the representations of unmasked tokens and the token embedding of [MASK]s, the decoder fuse all information together. In addition, the token embedding of [MASK] is in the same space as the latent representations of unmasked tokens.

3ML$_{cross}$ Both $3ML_{self}$ and $3ML_{cross}$ share the same encoder that only receives unmasked tokens as input. In contrast to $3ML_{self}$, we use a Transformer block with both self- & cross-attention as the base layer for the $3ML_{cross}$ decoder. Two sequences are given as input to the decoder, the latent representations of unmasked tokens and a sequence of [MASK] tokens and their position embeddings. The information flow from unmasked tokens to [MASK]s (2^{nd} flow) are conducted by the cross-attention module. The information among [MASK]s (3^{rd} flow) is transferred by the self-attention module. But there is no further information transfer among unmasked tokens (1^{st} flow) in the decoder, different from $3ML_{self}$. We use the same prediction layer for the randomly replaced, unchanged and masked tokens. Noticeably, we predict the randomly replaced and unchanged tokens after the encoder.

For both $3ML_{self}$ and $3ML_{cross}$, if the hidden dimensions of the encoder and decoder are not identical, one extra fully-connected layer is added

in between for projection. The prediction layer of vanilla MLM contains two fully-connected layers. The last one shares the weight from the embedding layer that has the same hidden dimension as the encoder. It projects the hidden dimension to the vocabulary size for prediction. If the hidden dimensions of the encoder and decoder are different, the first prediction layer projects the hidden dimension of the decoder back to the encoder dimension, so we can still share the weight from the embedding layer. More details are in Appendix D.

Fine-tuning and Inference After pre-training, the decoder of both architectures is removed. We only fine-tune the encoder on downstream tasks. We implement fine-tuning in this way because we want to: (1) speed up the inference; (2) keep our architecture for downstream tasks the same as standard MLMs and make it convenient for various applications without modifying their frameworks. However, for some tasks that require the [MASK] embedding, like the mask-infilling task, it might be beneficial to keep the decoder, since the token embedding of [MASK] doesn't locate in the same space as other tokens.

3 Experimental Setup

3.1 Tasks

We evaluate our pre-trained models on the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019b), which consists of 2 single-sentence classification tasks: CoLA (Warstadt et al., 2019) and SST (Socher et al., 2013), 3 similarity and paraphrase tasks: MRPC (Dolan and Brockett, 2005), QQP[3], and STS (Cer et al., 2017), and 4 natural language inference tasks: MNLI (Williams et al., 2018), QNLI (Rajpurkar et al., 2016), RTE (Dagan et al., 2005; Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009), and WNLI (Levesque et al., 2012). Like the original BERT paper (Devlin et al., 2019), we exclude WNLI, as the standard fine-tuning approach couldn't even beat the majority classifier.

We report accuracy for SST-2, MNLI, QNLI, and RTE, both F1 score and accuracy for MRPC and QQP, Matthew's correlation for CoLA, both Pearson and Spearman correlation for STS. By default, we use the same calculation as the GLUE leaderboard, i.e. the average of MNLI-m and MNLI-mm for MNLI, the average of F1 and accuracy for MRPC and QQP, and the average of Pearson and Spearman correlation for STS. We finally report the macro average of all tasks. For ablation experiments, we only evaluate the models on MNLI, QNLI, and QQP and report their accuracy, since these three tasks have the largest amount of training and validation sets, resulting in more stable fine-tuning scores than the others. In addition, the weighted average accuracy for MNLI from MNLI-m and MNLI-mm is shown rather than the macro average for ablation studies[4].

3.2 Baselines

We compare 3MLs to the following baselines:

- **Google BERT** The results of Google BERT (Devlin et al., 2019) on the GLUE development set are not shown in the original paper, we borrow the BERT-base's and BERT-large's results from Xu et al. (2020) and Clark et al. (2019), respectively.

- **Our 24hBERT** 24hBERT (Izsak et al., 2021) achieves comparable performance to the original BERT (Devlin et al., 2019) with an academically friendly computation budget (24 hours with 8 Nvidia Titan-V GPUs). Like Wettig et al. (2022), we re-implement it by adopting RoBERTa's BPE tokenizer (Sennrich et al., 2016; Liu et al., 2019) for better performance. This is the main baseline for our ablation studies.

- **Our RoBERTa** RoBERTa-base's GLUE results (trained on BooksCorpus and English Wikipedia) are not fully shown in the original paper (Liu et al., 2019). We re-implement it with its original hyperparameters.

- **ELECTRA** A discriminatively pretrained language model from Clark et al. (2020).

We don't include the encoder-decoder architectures (like BART (Lewis et al., 2020) and MASS (Song et al., 2019)) here because RoBERTa outperforms them (Lewis et al., 2020) on GLUE tasks.

3.3 Implementation.

The re-implementation of baselines and our pre-training methods are conducted on fairseq (Ott

[3] https://www.quora.com/profile/Ricky-Riche-2/First-Quora-Dataset-Release-Question-Pairs

[4] Both macro average and weighted average scores are comparable since MNLI-m and MNLI-mm contain a similar number of validation samples (9816 and 9833).

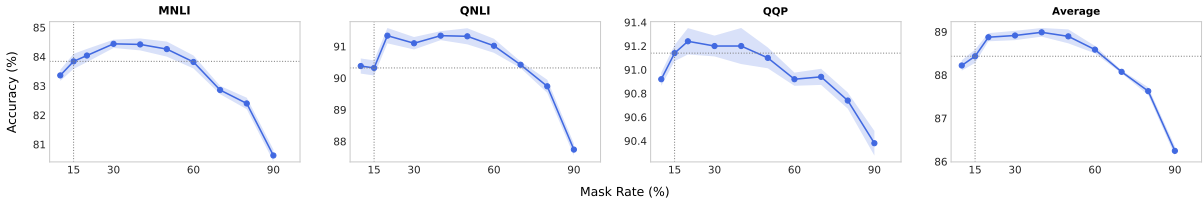

Figure 3: **Masking rate**. A middle-level masking rate (40%) works best on average.

et al., 2019). All results are from the models pre-trained on BooksCorpus and English Wikipedia that are tokenized by RoBERTa's BPE tokenizer with a vocabulary size of 50K[5]. In addition, all results with a single number are the median of five trials.

Training Recipes We have two pre-training recipes: an efficient recipe for a sequence length of 128 and a longer recipe for a sequence length of 512. The efficient pre-training recipe from 24hBERT (Izsak et al., 2021) is mainly used for ablation studies. It is a computation-friendly recipe that takes 9 hours with 16 Nvidia Tesla V100 GPUs. The longer pre-training recipe from RoBERTa (Liu et al., 2019) takes about 36 hours with 32 Nvidia Tesla V100 GPUs. With our efficient method, we can reduce the training time proportionally to the reduced computation (FLOPs). More hyper-parameter details for these two recipes are shown in Table 4 (see Appendix A). And the calculation details of training FLOPs are in Appendix D.

The default masking strategy for 3ML stays the same as BERT. That is for all corrupted tokens, 80% of them are replaced by [MASK], 10% are replaced by random tokens from the vocabulary and 10% stay unchanged. We borrow the same fine-tuning hyperparameters from 24hBERT for all 3MLs (Table 5 in Appendix B).

Architectures The encoder of our large or base model shares the same settings as Google BERT. By default, we use a two-layer decoder with half of the hidden dimension of the encoder. $3ML_{self}$ uses Transformer (Vaswani et al., 2017) encoder layers for its decoder, while $3ML_{cross}$ uses Transformer decoder layers with both self-attention and cross-attention layers without causal masking for its decoder. Since the hidden dimensions of the encoder and decoder are not the same, there is a linear layer in between to project the output from the encoder to the dimension of the decoder. 3MLs have two untied learnable positional embeddings for the encoder and decoder.

[5]This is the main reason for the different model parameters between 3ML and baselines. BERT and ELECTRA use a vocabulary with 30K tokens

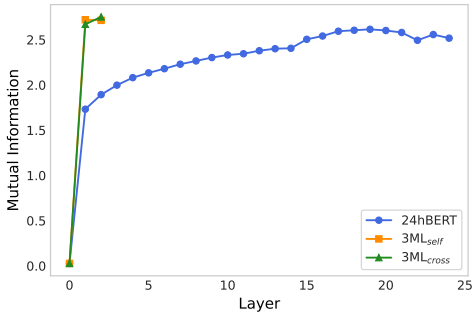

Figure 4: **Mutual Information** between hidden representations of [MASK] tokens per layer and the original tokens. Layer 0 corresponds to the token embeddings. All models are pre-trained with a masking rate of 40%.

Like 24hBERT, we implement pre-layer normalization (pre-LN) (Shoeybi et al., 2019) for 3ML-large. It makes the pre-training more stable and achieves better performance with a large learning rate. For $3ML_{self}$-base, post-LN is slightly better than pre-LN. We still use pre-LN for $3ML_{cross}$-base for stable pre-training. Post-LN doesn't work for $3ML_{cross}$. We leave the investigation of this problem to future work. For fine-tuning GLUE tasks, the 3ML decoder is removed. So its inference time on downstream tasks stays the same as the original BERT. More details of the 3ML architectures are in Table 4 (see Appendix A).

4 Two Prerequisites for Efficiency

In this section, we show that the two prerequisites, masking more and masking later, for efficiency hold for 3ML, which is also an empirical test of our information flow assumption.

4.1 Mask More

As shown in Figure 2, if masking more is possible, the sequence length of the input to 3ML's encoder becomes shorter. Since most trainable parameters are located in the encoder (the $3ML_{self}$-large encoder contains 98% of the parameters, excluding

Model	24hBERT	$3ML_{self}$	$3ML_{cross}$
PPL	7.8	9.5	9.3

Table 1: **MLM perplexity** of 24hBERT and 3MLs on the validation set. All models are pre-trained with a masking rate of 40%.

the embedding layer and the prediction layer) and the computation time of a self-attention module scales quadratically with the sequence length, this significantly reduces the computation time.

However, the motivation of our work is to maintain MLM's performance with higher efficiency. We don't want to lose performance significantly. Therefore, we conduct an experiment on a vanilla MLM, 24hBERT, to check whether masking more is possible.

As shown in Figure 3, the default masking rate (15%) from BERT is not optimal. With an increasing masking rate, the performance of all three tasks increases first and then decreases. The optimal masking rate is 30% for MNLI, 20% for QNLI, and 20% for QQP. The performance of a masking rate in (15, 45)% is consistently better than the one of 15%. On average, a masking rate of 40% works the best. Wettig et al. (2022) also obtained a similar result. In short, masking more is not only possible but also offers higher performance.

4.2 Mask Later

Masking later and masking more are complementary to achieve higher efficiency. For vanilla MLM, masking more offers better performance. But it doesn't guarantee that we can disentangle [MASK]s from the word embedding and append them at an intermediate layer. Instead of directly showing the performance of 3ML, we try to answer the following two questions for checking the possibility of masking later: (1) How good are 3MLs at masked language modeling compared to BERT? (2) How fast do the models recover the identity of the [MASK] tokens?

To answer the first question, we compare the MLM perplexity for [MASK]s between 24hBERT and 3MLs on the validation set. The results in Table 1 show: While $3ML_{self}$ and $3ML_{cross}$ have comparable perplexities, the perplexity of 24hBERT is significantly lower, indicating that 24hBERT performs better at the MLM task. However, this does not necessarily correlate with better performance on downstream tasks due to the mismatch between pre-training and fine-tuning (further discussion in §5.2). One could even argue that the increasing difficulty of the pre-training task forces 3MLs to learn better hidden representations of the unmasked tokens.

To address the second question, we measure the mutual information between the hidden representations at each layer at the masked positions and the original tokens at these positions. We follow the strategy proposed by Voita et al. (2019), and take hidden representations at masked positions corresponding to the 1000 most frequent tokens. For each layer, we gather 5M hidden representations and cluster them using mini-batch k-means into 10,000 clusters. For 24hBERT, we do this for each of the 24 encoder layers. For the 3ML models, the masked tokens are only fed in the decoder, values are only calculated for two decoder layers.

The results for 24hBERT and 3MLs are shown in Figure 4. For vanilla MLM, the largest amount of information on the identity of masked tokens is restored after the first few layers. The information is further gradually restored over the remaining layers. The results for $3ML_{self}$ and $3ML_{cross}$ are similar. After the first decoder layer, the information on the masked token identity is already restored to a similar level as in the last layer of 24hBERT. A possible explanation is that the decoder of 3MLs can already access the high-level representation from the encoder that facilitates the reconstruction.

The results from Figure 4 also empirically test our information flow assumption. The 1^{st} flow contains the most significant amount of information. We can achieve higher efficiency by specifically allocating more computation to this flow and spending less computation on the others, rather than focusing on all flows at once like vanilla MLM.

5 Efficient Setting

Section 4 shows that two prerequisites for higher efficiency hold for 3MLs. In this section, we further explore different choices of 3ML architecture and the masking rate, trying to select an optimal setting with the trade-off between performance and efficiency.

5.1 Decoder Architecture

By default, we set the 3ML's encoder with the same hyper-parameter setting as Google BERT. We leave the exploration of the encoder architecture to future work. Since both [MASK]s and unmasked tokens

#layer	3ML$_{self}$ MNLI	3ML$_{self}$ QNLI	3ML$_{self}$ QQP	3ML$_{cross}$ MNLI	3ML$_{cross}$ QNLI	3ML$_{cross}$ QQP
1	83.6	91.2	91.1	83.5	90.8	90.9
2	**84.0**	91.6	**91.2**	**83.6**	**91.3**	91.0
4	**84.0**	**91.7**	**91.2**	83.4	91.1	**91.1**
8	82.8	90.7	91.0	81.6	89.7	90.6

(a) **Decoder depth**. A shallow decoder with 2 or 4 layers performs better.

#dim	3ML$_{self}$ MNLI	3ML$_{self}$ QNLI	3ML$_{self}$ QQP	3ML$_{cross}$ MNLI	3ML$_{cross}$ QNLI	3ML$_{cross}$ QQP
256	83.8	91.3	91.0	83.0	90.9	**91.1**
512	84.0	**91.6**	**91.2**	**83.6**	**91.3**	91.0
768	**84.4**	91.5	**91.2**	83.3	90.7	90.9
1024	84.1	91.5	**91.2**	83.5	90.8	91.0

(b) **Decoder width**. A decoder with a small hidden dimension still performs well.

Masking Strategy (%) masked	replaced	unchanged	3ML$_{self}$ MNLI	QNLI	QQP	Avg.	PPL	3ML$_{cross}$ MNLI	QNLI	QQP	Avg.	PPL
80	10	10	84.0	**91.6**	91.2	88.9	9.5	**83.6**	**91.3**	**91.0**	**88.6**	9.3
100	0	0	83.1	90.7	90.9	88.2	15.6	82.4	90.3	**91.0**	87.9	15.1
80	20	0	82.5	90.4	91.0	87.9	14.0	83.1	90.7	90.9	88.2	**7.7**
80	0	20	**84.2**	**91.6**	**91.3**	**89.0**	**7.1**	82.9	90.7	90.9	88.2	14.8

(c) **Masking Strategy**. The masking strategies, 80-10-10 and 80-0-20, work better. PPL refers to the validation MLM perplexity.

Table 2: **3ML ablation experiments** on the large model. The default settings are marked in gray, i.e. 2 decoder layers, a hidden dimension of 512 for 3ML's decoder, and a masking rate of 40% with the 80-10-10 strategy.

are fed into the 3ML's decoder, it's necessary to select a small decoder for high efficiency.

We explore 3ML's decoders with different numbers of layers and hidden dimensions in Tables 2a and 2b. As shown in Table 2a, both 3ML$_{self}$ and 3ML$_{cross}$ have a similar but surprising finding: A large decoder with 8 layers works the worst, while a small decoder with 2 or 4 layers works the best. We argue that 3ML with a deeper decoder doesn't work well because of our fine-tuning setting. Recapping that 3ML's decoder is removed for fine-tuning, throwing a deeper decoder away means that more pre-trained parameters are removed. For the following experiments, 3ML with a two-layer decoder is the default setting.

Table 2b shows 3ML's performance with different hidden dimensions. 3ML is less sensitive to the hidden dimension of the decoder: the performance of all settings is very similar. By default, we choose a decoder with half of the encoder's dimension for the following experiments.

Both Tables 2a and 2b show that a small 3ML decoder is enough. It suggests that fusing all information flows after the encoder is easy, again empirically testing our information flow assumption.

5.2 Masking Strategy

The default masking strategy of vanilla MLM is 80-10-10. I.e. 80% of the corrupted tokens are replaced by [MASK]s, 10% are replaced by random tokens and 10% are kept unchanged. Ideally, we can achieve higher efficiency with the 100-0-0 setting, since we can further reduce the sequence length for 3ML's encoder given the same masking rate as 80-10-10.

We repeat the experiments on masking strategy as BERT in Table 2c, checking whether we have different findings for our 3ML. The default masking strategy 80-10-10 for both 3MLs works much better than 100-0-0. It is also the best strategy for 3ML$_{cross}$. 80-0-20 works similarly to 80-10-10 for 3ML$_{self}$. This finding suggests that keeping prediction on some original (unchanged) tokens is necessary, decreasing the gap between pre-training and fine-tuning. The original BERT paper (Devlin et al., 2019) had a similar finding. By default, we apply the 80-10-10 masking strategy.

We can further observe the mismatch between pre-training and fine-tuning with the MLM perplexity scores. The prediction of [MASK]s is the hardest for both 3MLs, with the highest perplexity. The prediction of unchanged tokens is the easiest for 3ML$_{self}$. Surprisingly, the prediction of randomly replaced tokens is the easiest for 3ML$_{cross}$, which is contradictory to our intuition that unchanged tokens should be easiest to predict.

5.3 Masking Rate

A large masking rate is a necessary prerequisite for the efficiency of 3ML. However, a too-large masking rate hurts the performance as shown in Figure 3. In this section, we study 3ML's trade-off between efficiency and performance. From Figure 5, we achieve higher speedups with increasing masking rates. With a masking rate of 50%, we can obtain near 1.5 times speedup, saving 1/3 of the computation budget.

Both 3ML$_{self}$ and 3ML$_{cross}$ share a similar trend as 24hBERT: better accuracy is obtained with a

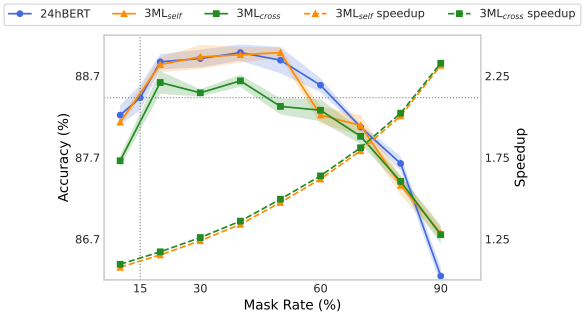

Figure 5: **Efficiency *vs.* performance**. The best results for 3ML$_{\text{self}}$ and 3ML$_{\text{cross}}$ are obtained with a masking rate of 50% and 40%, respectively. The accuracy is the macro average accuracy for MNLI, QNLI, and QQP. Speedup is calculated based on the pre-training FLOPs with 24hBERT as the baseline. Experiments are conducted on the large model.

middle-level masking rate. 3ML$_{\text{self}}$ works almost the same as 24hBERT with a masking rate of $< 40\%$. A masking rate of 50% works best for 3ML$_{\text{self}}$, even better than the best 24hBERT. It means that we can save 1/3 of the original training budget without any performance drop and even with slight improvement. 3ML$_{\text{cross}}$ works slightly worse than 24hBERT with the same masking rate. It performs best with a masking rate of 40% and better than 24hBERT with the default masking rate (15%). The training FLOPs of 24hBERT are 1.37 times larger than the ones of 3ML$_{\text{cross}}$ when the masking rate is 40%. We recommend choosing a masking rate of 40% or 50% for both 3MLs for obtaining good performance and high efficiency.

6 Comparison with Previous Work

In this section, we compare 3ML with strong baselines on the development set of GLUE. We list the results of both efficient and longer pre-training recipes in Table 3.

6.1 Result of Efficient Pre-training Recipe

The first block of Table 3 shows results from the efficient pre-training recipe. We train all models with the same number of updates and an identical learning rate. So the improvement shown here is not obtained by seeing more data or doing more extensive gradient updates.

Similar to Figure 3, 24hBERT with a masking rate of 40% performs consistently better than the one with 15% on all GLUE tasks, achieving 0.7% absolute improvement on average. It further suggests that masking more is possible and favorable. 3ML$_{\text{self}}$ with a masking rate of 40% and 50% has the same average score as 24hBERT-40%. But it requires less computation, to be precise only 75% and 68% of the original computation budget. 3ML$_{\text{cross}}$ achieves a slightly worse result than 24hBERT-40%, losing 0.2% performance on average, but being slightly more efficient than 3ML$_{\text{self}}$ with the same masking rate.

Compared to the number of trainable parameters of 24hBERT, the increasing parameters for 3ML due to an extra decoder are negligible, only accounting for 2% of all 24hBERT parameters. In addition, 3ML's decoder is discarded for fine-tuning. We believe that the well-performed 3MLs benefit from our two-stage training method rather than the slightly added parameters.

6.2 Result of Longer Pre-training Recipe

3ML behaves well and efficiently on a limited computation budget. We are also interested in its scaling behavior which is critical for a language model. As we normally train a language model on large data for better generalization on a wide range of tasks. Due to our limited computation resources, we only implement the scaling experiment on the base model with more updates till convergence. We leave the scaling experiments on a bigger data set and a larger model to future work.

The results from the longer pre-training recipe are shown in the second block of Table 3. When seeing the same amount of data (125K updates), 3ML$_{\text{self}}$ performs better than BERT (84.9/85.0 vs. 82.5), comparable to RoBERTa (84.9/85.0 vs. 84.9) and ELECTRA (84.9/85.0 vs. 85.1). However, 3ML-125K is faster than any baseline (28% faster than RoBERTa with a masking rate of 50%). When using the same computation budget as RoBERTa, 3ML$_{\text{self}}$-40%-153K achieves the best result (85.3) among all models. We also notice that the pre-training of 3ML$_{\text{self}}$s already converge with 125K updates. As also shown in Table 3, we only obtain 0.4% and 0.5% improvement with extra 28K and 35K updates for 3ML$_{\text{self}}$-40% and 3ML$_{\text{self}}$-50%, respectively. Further improvement is expected if the model is trained on a larger dataset.

Similar to the efficient pre-training recipe, 3ML$_{\text{cross}}$ performs worse than the strong baselines, RoBERTa and ELECTRA, but better than BERT-base. We argue the reason for the poor performance of 3ML$_{\text{cross}}$ is: The positional embeddings for [MASK]s and unmasked tokens are not in the

Model	#Params M	Speedup	CoLA Mcc	SST Acc	MRPC F1/Acc	STS Pear/Spea	QQP F1/Acc	MNLI-m/mm Acc	QNLI Acc	RTE Acc	Avg.
Efficient pre-training recipe, large model											
24hBERT-15%*	355	1.00×	61.3	93.0	92.0/89.0	89.4/89.2	88.0/91.1	84.0/83.6	90.4	76.9	84.3
24hBERT-40%*	355	1.00×	61.4	**93.2**	**92.1/89.2**	**89.5/89.5**	88.0/91.2	**84.4/84.3**	91.3	80.1	**85.0**
$3ML_{self}$-40%	362	1.34×	61.5	92.3	92.1/89.0	**89.5/89.5**	**88.1/91.2**	**84.4/84.3**	**91.6**	**80.9**	**85.0**
$3ML_{self}$-50%	362	1.47×	**62.4**	92.3	92.0/89.2	**89.5/89.5**	**88.1/91.2**	84.1/84.4	91.4	79.8	**85.0**
$3ML_{cross}$-40%	364	1.37×	**62.4**	92.8	91.2/87.5	89.0/88.8	87.9/91.0	83.7/83.5	91.3	80.5	84.8
Longer pre-training recipe, base model											
BERT	110	1.15×	54.3	91.5	89.5	88.9	89.8	83.5	91.2	71.1	82.5
RoBERTa*	125	1.00×	61.3	**93.5**	91.5/88.8	89.7/89.5	**88.1/91.3**	84.7/84.9	91.4	78.9	84.9
ELECTRA	126	1.15×	-	-	-/-°	-/-°	-/-°	-	-	-	85.1
$3ML_{self}$-40%-125K◇	129	1.22×	60.6	92.9	**93.2/90.7**	89.9/89.7	87.7/90.9	84.5/84.3	91.3	78.7	84.9
$3ML_{self}$-40%-153K◇	129	1.00×	61.4	92.9	92.3/89.2	**90.3/90.1**	87.9/91.1	**84.8/84.9**	91.4	**81.2**	**85.3**
$3ML_{self}$-50%-125K◇	129	1.28×	**63.1**	92.4	92.0/89.0	89.7/89.4	87.6/90.9	84.1/84.3	91.9	79.1	85.0
$3ML_{self}$-50%-160K◇	129	1.00×	61.7	93.3	92.9/90.2	90.1/89.9	87.9/91.1	84.3/84.5	91.3	79.1	85.1
$3ML_{cross}$-40%-157K◇	130	1.00×	56.2	92.3	91.8/88.7	89.6/89.3	87.6/90.8	82.7/83.6	90.8	78.3	83.7

Table 3: **Complete results on GLUE dev. set**. Speedup is computed based on the pre-training FLOPs. "*" denotes our re-implementation. "°" means the corresponding metric is used for calculating the average score. When using the same metrics as ELECTRA to compute the average score, models with "◇" stay the same as the shown average score. More details on the baseline models are in Appendix C.

same latent space (Figure 2), which makes it difficult for the model to fuse all information and predict missing information. In addition, the decoder doesn't further transfer information among unmasked tokens (the most complex flow). We leave the further investigation to future work.

In summary, when trained on the same number of samples, $3ML_{self}$ performs similarly to strong baselines with less computation. $3ML_{cross}$ performs comparably to the baselines for efficient pre-training. When trained with a similar amount of computation, $3ML_{self}$ performs the best and $3ML_{cross}$ performs better than the standard baseline, BERT.

7 Related Work

The most related work to this paper is MAE (He et al., 2021) from computer vision. $3ML_{self}$ shares almost the same architecture as MAE, but with the additional prediction of unchanged tokens and randomly replaced tokens, while MAE only reconstructs the masked patches. We also have different findings from MAE: A small decoder works better for $3ML_{self}$. While MAE applied a deeper decoder for better performance.

3ML's encoder-decoder architecture looks similar to BART (Lewis et al., 2020) and MASS (Song et al., 2019). But BART and MASS apply a causal masking decoder that is suitable for generation tasks rather than classification tasks. Our decoder is still a bidirectional architecture like the encoder. In addition, the decoder of BART and MASS is used for fine-tuning downstream tasks. For GLUE tasks, one needs to input the same sequence to both encoder and decoder, which is less efficient. 3ML's decoder is removed for fine-tuning, having the same inference speed as vanilla MLM.

Hou et al. (2022) do a concurrent work, dropping the representations of unimportant tokens for some intermediate layers to reduce the sequence length for efficiency. We don't include any prior information and only drop the [MASK] token at the very beginning. In addition, 3ML achieves better results.

Wettig et al. (2022) and our work have the same finding, better performance for a middle-level masking rate. Although ALBERT (Lan et al., 2020) and ELECTRA (Clark et al., 2020) make the pre-training of MLM more efficient, their studies are orthogonal to ours.

8 Conclusion

We propose a two-stage learning method for efficient masked language modeling and design two models, $3ML_{self}$ and $3ML_{cross}$, for our method. Two prerequisites for our efficient method are: We can have a higher masking rate and append [MASK]s at a later layer. Our experiments show that both, masking more and masking later, are possible and favorable. This allows us to reduce the sequence length of the encoder during pre-training by a factor depending on the masking rate. By conducting extensive experiments, we observe that $3ML_{self}$ performs better on downstream tasks than $3ML_{cross}$. It can speed up the pre-training by a factor of 1.5x for our efficient pre-training recipe without any performance degradation. Using roughly the same computation budget, $3ML_{self}$ outperforms all of our strong baselines like ELECTRA and RoBERTa.

Limitations

Our investigation is limited to classification tasks. While 3ML outperforms other models on GLUE tasks, it might not be good at other tasks, especially for the mask-infilling task where the token embedding of [MASK] is used directly. More tasks need to be evaluated to validate 3ML's robustness.

In addition, we only train 3MLs on BookCorpus and English Wikipedia. The scaling behavior of 3ML with respect to model size and amount of data is an open question. Further, it needs to be validated whether the results transfer to other languages than English. We leave this to the future. We also don't apply any fine-tuning tricks, like layer-wise learning rate in ELECTRA, and tune the hyperparameters. It would be better for us to provide a specific optimal recipe for our model, making it more practical.

Acknowledgments

The authors from RWTH Aachen University partially received funding from eBay Inc. We thank the colleagues from eBay (Michael Kozielski and Shahram Khadivi) and RWTH Aachen University (Yingbo Gao and Christian Herold) for their discussion.

References

Luisa Bentivogli, Bernardo Magnini, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2009. The fifth PASCAL recognizing textual entailment challenge. In *Proceedings of the Second Text Analysis Conference, TAC 2009, Gaithersburg, Maryland, USA, November 16-17, 2009*. NIST.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. 2017. SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. pages 1–14.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. 2019. BAM! born-again multi-task networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5931–5937, Florence, Italy. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The PASCAL recognising textual entailment challenge. In *Machine Learning Challenges, Evaluating Predictive Uncertainty, Visual Object Classification and Recognizing Textual Entailment, First PASCAL Machine Learning Challenges Workshop, MLCW 2005, Southampton, UK, April 11-13, 2005, Revised Selected Papers*, volume 3944 of *Lecture Notes in Computer Science*, pages 177–190. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing, IWP@IJCNLP 2005, Jeju Island, Korea, October 2005, 2005*. Asian Federation of Natural Language Processing.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third PASCAL recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL@ACL 2007 Workshop on Textual Entailment and Paraphrasing, Prague, Czech Republic, June 28-29, 2007*, pages 1–9. Association for Computational Linguistics.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Efficient training of BERT by progressively stacking. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2337–2346. PMLR.

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7, Florence, Italy.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. 2021. Masked autoencoders are scalable vision learners. *CoRR*, abs/2111.06377.

Le Hou, Richard Yuanzhe Pang, Tianyi Zhou, Yuexin Wu, Xinying Song, Xiaodan Song, and Denny Zhou. 2022. Token dropping for efficient BERT pretraining. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3774–3784, Dublin, Ireland. Association for Computational Linguistics.

Peter Izsak, Moshe Berchansky, and Omer Levy. 2021. How to train BERT with an academic budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10644–10652, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012*. AAAI Press.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7871–7880. Association for Computational Linguistics.

Yang Liu. 2019. Fine-tune BERT for extractive summarization. *CoRR*, abs/1903.10318.

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. *Trans. Assoc. Comput. Linguistics*, 8:726–742.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, abs/1907.11692.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pages 48–53. Association for Computational Linguistics.

Zizheng Pan, Bohan Zhuang, Jing Liu, Haoyu He, and Jianfei Cai. 2021. Scalable vision transformers with hierarchical pooling. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 367–376. IEEE.

David A. Patterson, Joseph Gonzalez, Quoc V. Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David R. So, Maud Texier, and Jeff Dean. 2021. Carbon emissions and large neural network training. *CoRR*, abs/2104.10350.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1631–1642. ACL.

Amir Soleimani, Christof Monz, and Marcel Worring. 2020. BERT for evidence retrieval and claim verification. In *Advances in Information Retrieval - 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14-17, 2020, Proceedings, Part II*, volume 12036 of *Lecture Notes in Computer Science*, pages 359–366. Springer.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: masked sequence to sequence pre-training for language generation. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5926–5936. PMLR.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3645–3650. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz

Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Elena Voita, Rico Sennrich, and Ivan Titov. 2019. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. 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 4396–4406, Hong Kong, China. Association for Computational Linguistics.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3261–3275.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Trans. Assoc. Comput. Linguistics*, 7:625–641.

Alexander Wettig, Tianyu Gao, Zexuan Zhong, and Danqi Chen. 2022. Should you mask 15% in masked language modeling? *CoRR*, abs/2202.08005.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Qiyu Wu, Chen Xing, Yatao Li, Guolin Ke, Di He, and Tie-Yan Liu. 2021. Taking notes on the fly helps language pre-training. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. BERT-of-theseus: Compressing BERT by progressive module replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869, Online. Association for Computational Linguistics.

Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, Wengang Zhou, Houqiang Li, and Tie-Yan Liu. 2020. Incorporating BERT into neural machine translation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

A Pre-training Hyperparameters

Pre-training hyperparameters are shown in Table 4. Settings for large and base models are mainly borrowed from 24hBERT (Izsak et al., 2021) and RoBERTa (Liu et al., 2019), respectively.

B Fine-tuning Hyperparameters

Fine-tuning hyperparameters for GLUE are shown in Table 5, borrowed from 24hBERT (Izsak et al., 2021). For each task, we run the fine-tuning on the whole search space five times with different random seeds, then select the best values from each running and finally choose the median of these five values.

C Differences among Baselines

We use some base models (BERT, RoBERTa, and ELECTRA) as our baselines for the longer pre-training recipe. We specify the main differences among them here. RoBERTa contains 15M more trainable parameters (see Table 3) than BERT because they apply different sub-word algorithms and have different vocabulary sizes, 50K for RoBERTa, and 30K for BERT (also for ELECTRA). The extra 16M parameters (compared to BERT) from ELECTRA come from the generator, while its discriminator shares the same architecture as BERT. In addition, BERT, ELECTRA, and RoBERTa are trained with a batch size of 256, 256, and 2048, respectively. Their number of updates is 1M, 766K, and 125K, respectively. In another word, BERT and RoBERTa see the same amount of data (256M samples), while ELECTRA sees less data (197M samples). But RoBERTa conducts the least updates. Among these three models, the training recipe from RoBERTa is more suitable for scaling. We can train an MLM quickly with a larger batch size by allocating a large number of GPUs. Therefore, we borrow

Hyperparameter	3ML-large	3ML-base
Encoder		
Number of Layers	24	12
Hidden Size	1024	768
FFN Inner Hidden Size	4096	3072
Attention Heads	16	12
Attention Head Size	64	64
Decoder		
Number of Layers	2	2
Hidden Size	512	384
FFN Inner Hidden Size	2048	1536
Attention Heads	8	6
Attention Head Size	64	64
Whole model		
Dropout	0.1	0.1
Attention Dropout	0.1	0.1
Layer Normalization	pre-LN	post-LN $3ML_{self}$ / pre-LN $3ML_{cross}$
Sequence Length	128	512
Optimizer		
Warmup Proportion	0.06	0.06
Peak Learning Rate	2e-3	7e-4
Batch Size	4096	2048
Weight Decay	0.01	0.01
Max Steps	23K	153K for $3ML_{self}$ / 160K $3ML_{cross}$
Learning Rate Decay	Linear	Linear
Adam ϵ	1e-6	1e-6
Adam (β_1, β_2)	(0.9, 0.98)	(0.9, 0.98)
Gradient Clipping	0.0	0.0

Table 4: **Pre-training hyperparameters** for both $3ML_{self}$ and $3ML_{cross}$. Pre-LN $3ML_{cross}$ is more stable than Post-LN. If the gradient explodes, set gradient clipping as 1.0 instead.

Hyperparameter	QQP, MNLI, QNLI	CoLA, SST, MRPC, STS, RTE
Learning Rate	{5e-5, 8e-5}	{1e-5, 3e-5, 5e-5, 8e-5}
Batch Size	32	{16, 32}
Weight Decay	0.1	0.1
Max Epochs	{3, 5}	{3, 5, 10}
Warmup Proportion	0.06	0.06

Table 5: **Fine-tuning hyperparameters** for both $3ML_{self}$ and $3ML_{cross}$. Same as RoBERTa (Liu et al., 2019), RTE, STS, and MRPC are fine-tuned from the MNLI model instead of the pre-trained model.

the training recipe from RoBERTa as our longer pre-training recipe.

D Calculation of FLOPs

We borrow a simplified version of FLOPs calculation from Pan et al. (2021) where the computation for bias, activation, and dropout is neglected because it only occupies a small amount (< 1%) of the total FLOPs. We restate their calculation and make some modifications here. The meaning for different notations is listed in Table 6 for your convenience.

Transformer block with self-attention Given the sequence length n and the embedding dimension d, the FLOPs of the multi-head self-attention (MSA) layer come from: (1) the projection of the input sequence to key, query and, value $\phi_{qkv} = 2 \cdot 3nd^2$ [6]; (2) the attention map from key and query $\phi_{map} = 2n^2d$; (3) the self-attention operation $\phi_{attn} = 2n^2d$; (4) the projection of the self-attention output $\phi_{out} = 2n^2d$. Then the overall FLOPs for an MSA layer are:

$$\phi_{MSA}(n, d) = 8nd^2 + 4n^2d$$

There are two fully-connected (FC) layers for a feed-forward (MLP) layer. The first FC projects the embedding dimension from d to $4d$, and the second one project it back to d. The FLOPs are:

$$\phi_{MLP}(n, d) = 2 \cdot 8nd^2 = 16nd^2$$

So the overall FLOPs for a transformer block are:

$$\begin{aligned}\phi_{BLK}(n, d) &= \phi_{MSA}(n, d) + \phi_{MLP}(n, d) \\ &= 24nd^2 + 4n^2d\end{aligned}$$

Embedding layer We assume all models, including baseline models, implement sparse lookup for token embedding, which is different from ELECTRA (Clark et al., 2020) which states RoBERTa and BERT obtain the token embedding by multiplying the embedding layer with one-hot vectors. We make this assumption since any model can be easily re-implemented in this efficient way. Sparse lookup is efficient. It can be neglected for calculating FLOPs.

Prediction layer The prediction layer consists of two FC layers. The first FC layer keeps the input and output sequence in the same dimension, and the second one projects to the vocabulary size $|V|$. Like typical language models, we use shared weight between the embedding layer and the second FC layer. Like RoBERTa, we only input the masked tokens, including unchanged and randomly replaced tokens, to the prediction layer. With a masking rate of r, the FLOPs are:

$$\phi_{Pred}(n, r, d, |V|) = 2nr(d^2 + d|V|)$$

RoBERTa The total FLOPs for RoBERTa can be computed as

$$\begin{aligned}&\phi_{RoBERTa}(b, u, l, n, d, r, |V|) \\ =&2bu(l \cdot \phi_{BLK}(n, d) + \phi_{Pred}(n, r, d, |V|)) \quad (2)\end{aligned}$$

where b is the batch size, u is the number of updates, l is the number of transformer blocks and 2 at the beginning denotes the forward and backward process. Both forward and backward processes consume similar FLOPs.

3ML$_{\text{self}}$ has two types of transformer blocks, one for the encoder and one for the decoder. The only difference between them is the hidden dimension. To project the output sequence from the encoder to the same hidden dimension of the decoder, we add an FC layer in between. The FLOPs of the prediction layer are modified as:

$$\begin{aligned}&\phi'_{Pred}(n, r, d_{en}, d_{de}, |V|) \\ =&2nr(d_{de}d_{en} + d_{en}|V|)\end{aligned}$$

where d_{en} and d_{de} is the hidden dimension of the encoder and decoder, respectively. Then the overall FLOPs are:

$$\begin{aligned}&\phi_{3ML_{\text{self}}}(b, u, l_{en}, l_{de}, n_{en}, n, d_{en}, d_{de}, r, |V|) \\ =&2bu(2n_{en}d_{en}d_{de} + l_{en} \cdot \phi_{BLK}(n_{en}, d_{en}) \\ &+ l_{de} \cdot \phi_{BLK}(n, d_{de}) \\ &+ \phi'_{Pred}(n, r, d_{en}, d_{de}, |V|)) \quad (3)\end{aligned}$$

where $n_{en} = (1 - 0.8r)n$ since only 80% of all masked tokens are replaced by [MASK]s. The first term on the right is for the dimension projection from the encoder to the decoder. l_{en} and l_{de} are the number of Transformer blocks for the encoder and decoder, respectively.

Transformer block with self & cross-attention The decoder of 3ML$_{\text{cross}}$ contains an MSA layer, a multi-head cross-attention (MCA) layer, and an MLP layer. The FLOPs calculation of both MSA and MLP stay the same as above. Similar to the

[6] We double all original calculation by considering the fused multiply-add ops (Clark et al., 2020).

MSA layer, the FLOPs for different components of the MCA layer are

$$\begin{aligned}\phi^{'}_{qkv} &= 4n_{en}d_{de}^2 + 2n_{de}d_{de}^2\\ \phi^{'}_{map} &= 2n_{en}n_{de}d_{de}\\ \phi^{'}_{attn} &= 2n_{en}n_{de}d_{de}\\ \phi^{'}_{out} &= 2n_{de}d_{de}^2\end{aligned}$$

where n_{en} and n_{de} are the sequence lengths of the encoder and decoder inputs, respectively. So the FLOPs for an MCA layer are:

$$\begin{aligned}&\phi_{MCA}(n_{en}, n_{de}, d_{de})\\ =&4n_{en}d_{de}^2 + 4n_{de}d_{de}^2 + 4n_{en}n_{de}d_{de}\end{aligned}$$

Then the overall FLOPs for a cross-attention Transformer block are:

$$\begin{aligned}&\phi_{CBLK}(n_{en}, n_{de}, d_{de})\\ =&\phi_{BLK}(n_{de}, d_{de}) + \phi_{MCA}(n_{en}, n_{de}, d_{de})\\ =&4n_{en}d_{de}^2 + 28n_{de}d_{de}^2 + 4n_{en}n_{de}d_{de} + 4n_{de}^2d_{de}\end{aligned}$$

3ML$_{\text{cross}}$ Similar to 3ML$_{\text{self}}$, 3ML$_{\text{cross}}$ has an encoder and a decoder with a hidden size of d_{en} and d_{de}, respectively. Since their hidden sizes might be different, there is an FC layer projecting from d_{en} to d_{de}. Then the overall FLOPs for training a 3ML$_{\text{cross}}$ is

$$\begin{aligned}&\phi_{3ML_{\text{cross}}}(b, u, l_{en}, l_{de}, n_{en}, n_{de}, d_{en}, d_{de}, |V|)\\ =&2bu(2n_{en}d_{en}d_{de} + l_{en}\cdot\phi_{BLK}(n_{en}, d_{en})\\ &+ l_{de}\cdot\phi_{CBLK}(n_{en}, n_{de}, d_{de})\\ &+ \phi_{Pred}(n_{en}, \frac{0.2r}{1-0.8r}, d_{en}, |V|))\\ &+ \phi^{'}_{Pred}(n_{de}, 1, d_{en}, d_{de}, |V|))\end{aligned} \quad (4)$$

where $n_{de} = 0.8nr$ and $n_{de} = (1-0.8r)n$, since only 80% of all masked tokens are replaced by [MASK]s. The second last term denotes the prediction layer on the encoder side. Both prediction layers share the weight from the embedding layer. The masking rate from the second last term is achieved by calculating the ratio between the number of unchanged and randomly replaced tokens and n_{en}. The masking rate of the last term is 1 because there are only [MASK]s on the decoder side.

Notation	Explanatiorebib
n	sequence length
n_{en}	sequence length for encoder
n_{de}	sequence length for decoder
d	hidden dimension
d_{en}	hidden dimension for encoder
d_{de}	hidden dimension for decoder
r	masking rate
$\|V\|$	vocabulary size
b	batch size
u	#updates
l	#(Transformer block)
l_{en}	#(Transformer block) for encoder
l_{en}	#(Transformer block) for decoder

Table 6: Notation for the calculation of training FLOPs.