Effective Pre-Training Objectives for Transformer-based Autoencoders

Luca Di Liello[1]*, **Matteo Gabburo**[1]*, **Alessandro Moschitti**[2]
[1]University of Trento, [2]Amazon Alexa AI
{luca.diliello,matteo.gabburo}@unitn.it
{amosch}@amazon.com

Abstract

In this paper, we study trade-offs between efficiency, cost and accuracy when pre-training Transformer encoders with different pre-training objectives. For this purpose, we analyze features of common objectives and combine them to create new effective pre-training approaches. Specifically, we designed light token generators based on a straightforward statistical approach, which can replace ELECTRA computationally heavy generators, thus highly reducing cost. Our experiments also show that (i) there are more efficient alternatives to BERT's MLM, and (ii) it is possible to efficiently pre-train Transformer-based models using lighter generators without a significant drop in performance.

1 Introduction

Transformer-based models (Vaswani et al., 2017) require expensive hardware to be pre-trained (Strubell et al., 2019; Brown et al., 2020). Recently, many works focused on reducing pre-training cost (Lan et al., 2020; Sanh et al., 2020; Turc et al., 2019). ELECTRA, for example, proposes to train BERT as a discriminator rather than a generator (Clark et al., 2020). They replace the Masked Language Modeling objective (MLM) (Devlin et al., 2019) with Token Detection (TD). Then, the discriminator detects if input tokens are original or fake created by a small generator network.

On the one hand, the discriminator is much more efficient than MLM. On the other hand, the use of a generator requires the pre-training of a second transformer, increasing the pre-training cost. ELECTRA has been shown to be more accurate than BERT. However, it is not clear if this superior performance is due to its innovative architecture or to the long and extensive training, which highly increases the computation cost for obtaining the final language model.

In this paper, we study pre-training strategies with respect to the trade-off between efficiency, cost, and accuracy. Theoretical efficiency and computational cost do not always align well because the latter is influenced by the underlying infrastructure and by hardware acceleration technologies (i.e., NVIDIA Tensor Cores). For this purpose, we analyze the main important components of pre-training, i.e., pre-training objectives and the algorithms with which they are applied. For example, we note that MLM needs a large classification head that spans over the whole vocabulary (which usually contains several tens of thousands of tokens), while TD requires a smaller head, which is much more efficient and uses low computation resources.

We summarize our contribution as follows: First, we propose Random Token Substitution (RTS) and Cluster-based Random Token Substitution (C-RTS), two fast alternatives to ELECTRA's generator, which allows us to set a middle-ground in the trade-off between efficiency and accuracy. Indeed, RTS consists in just detecting tokens that are randomly changed into others, so very low cost, while C-RTS, which is a bit more expensive than RTS, exploits the knowledge about predictions in previous iterations to select more challenging replacements. Both our objectives increase the efficiency (20% - 45%) thanks to a much smaller binary classification head on top and are equally accurate to MLM on most of the tasks from a statistical significance viewpoint. We also demonstrate that, if trained for a longer time, C-RTS outperforms RTS on many benchmarks because it is a more challenging pre-training task.

Second, we propose Swapped Language Modeling (SLM), a variant of BERT's MLM that only replaces tokens with others, thus removing the special MASK token, which is responsible for BERT's pre-training/fine-tuning discrepancy (Clark et al., 2020). We show that this objective increases cost with respect to RTS and C-RTS, but outperforms

*Equal contribution

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 5533–5547
December 7-11, 2022

MLM in almost every task using precisely the same computational budget.

Finally, we empirically study the trade-offs mentioned above by pre-training standard BERT models with the proposed objectives, also comparing them with state-of-the-art architectures trained and tested on the same data. We perform an accurate comparison by evaluating our models on several natural language inference benchmarks: all tasks in the GLUE benchmark suite, ASNQ, WikiQA and TREC-QA, reporting accuracy as well as efficiency and cost. To better assess the latter, we also test the impact of objectives on smaller architectures (e.g., BERT-*small*), showing that our approaches have a broader impact on those classes of models.

2 Related Work

Many different objectives for self-supervised learning have been proposed in recent works, such as Causal Language Modeling (CLM) (Radford, 2018; Radford et al., 2019; Brown et al., 2020), Masked Language Modeling (MLM) (Devlin et al., 2019; Liu et al., 2019b) and Token Detection (TD) (Clark et al., 2020), the latter used by ELECTRA, which is composed by a generator and a discriminator. While the generator is trained with MLM to find suitable candidates to replace the special MASK tokens, the discriminator should recognize the replacements in the original text instead. After pre-training, the generator is removed, and the discriminator is used as the pre-trained language model. ELECTRA introduces many innovations: (i) the exploitation of the whole output of the discriminator to compute the loss function, thus having a stronger signal for the back-propagation; (ii) the usage of a generator network to find suitable replacements and (iii) the fact that the discriminator does not see spurious tokens such as the MASK token. The latter is a main drawback of the original BERT, as it creates input discrepancies between pre-training and fine-tuning, since the CLS representation is dependent on all input tokens thanks to the self-attention mechanism.

Other research directions to reduce training time address the architecture instead of the learning objective. In ALBERT (Lan et al., 2020), the authors tie the weights of every Transformer layer to save GPU memory, thus enabling bigger batch sizes. However, since the expressive power of their models is reduced when layers are tied, they must train for much longer. Sanh et al. (2020) and Turc et al. (2019) instead use distillation to reduce the model size, but the pre-training is still expensive because it requires a large teacher architecture.

Although pre-training is performed only once, it usually requires weeks and costly machines (Liu et al., 2019b; Brown et al., 2020), so it is important to find alternative ways to pre-train transformers. Tay et al. (2020) provides an overview of many recent advancements in transformer efficiency.

Another successful MLM improvement regarding the objectives is SpanBERT (Joshi et al., 2020), which proposes two new objectives: Span-Masking and Span-Boundary-Objective (SBO). Specifically, the Span-Masking objective is a refined version of MLM that masks contiguous spans of text instead of single tokens, while, with SBO, they predict the span content by considering only the output representation corresponding to the tokens on the boundaries. Furthermore, in Zhang et al. (2020), the authors propose a technique to improve downstream performance by adapting the model to the final task while pre-training. Similarly, in Di Liello et al. (2022) they do continuous pre-training with custom objectives to better adapt the model for Answer Sentence Selection (AS2).

In T5 (Raffel et al., 2020), the authors propose to use deshuffling (Liu et al., 2019a) to pre-train an autoregressive model. They shuffle random spans of text and ask the model to output tokens in the original order. This technique provides good results on an extensive collection of benchmarks. However, we cannot compare with them because we focus on autoencoder architectures only.

Finally, we mention the work by Izsak et al. (2021), in which the authors list many optimizations that could be applied to the transformers for faster pre-training. They also claim that using larger models with the same runtime leads to better results. We focus instead on the pre-training objective efficiency and on the classification head size. Thus we state that those techniques are orthogonal to our work and could be applied along with our alternative pre-training objectives.

3 Background on Pre-training Objectives

Before describing our models, we provide a detailed description of the most common token-level pre-training objectives used in the literature.

Masked Language Model (MLM) was proposed by Devlin et al. (2019). In MLM, 15% of the input tokens are replaced with a special mask,

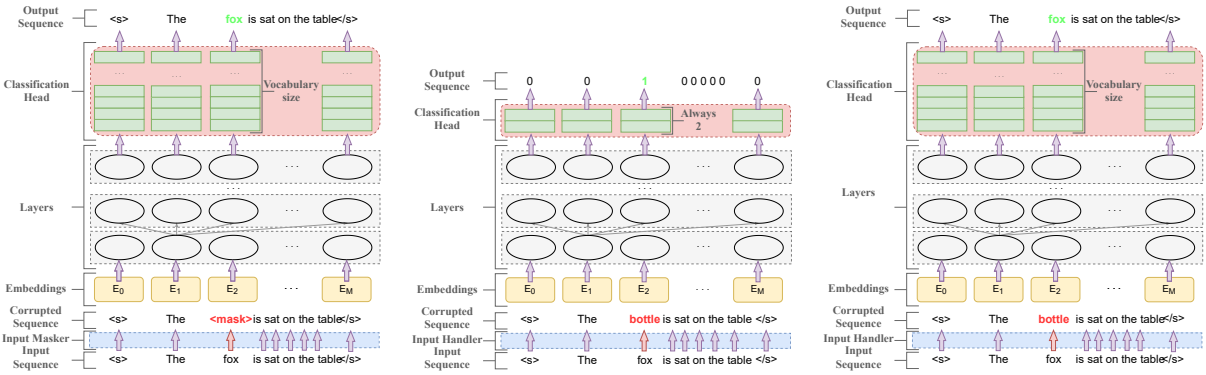

Figure 1: MLM, RTS and SLM architectures (from left to right). Notice that the classification head used by RTS is several times smaller than those used by MLM and SLM (see Appendix G).

and the model has to predict the original value. An improvement of MLM is *whole-word-masking*, in which masking is applied to every token belonging to a word and not just to independent tokens.

The model needs a large classification head for pre-training, as shown in Figure 1. Its dimension is proportional to the vocabulary size, and (especially for *small* models) this constitutes a significant fraction of the entire architecture parameters. In the *base* architectures, the MLM head constitutes about 20% of the model parameters while for *small* models, the fraction increases to 30%[1]. The memory footprint of the LM head while training is about 47% for *base* and 64% for *small* models[2]. See Appendix G for more details. For this reason, a binary classification head (as for TD) can provide significant efficiency improvement. Moreover, sharing the parameters of the MLM classification head with the embeddings does not reduce the computational cost but leads only to marginally lower memory requirements. In the embedding layer, only a few row vectors corresponding to the actual sentence tokens are updated at every step. On the contrary, MLM's softmax continuously computes gradients for the whole output linear transformation.

Causal Language Model (CLM) is used to train autoregressive models by predicting the next token in a sequence (Radford et al., 2019; Radford, 2018). Similarly to MLM, it requires a large classification head to output predictions over the whole vocabulary.

Permutation Language Modeling (PLM) was proposed by Yang et al. (2020) to combine the generative power of autoregressive models with the bidirectional context of autoencoders. This is accomplished by permuting the input tokens and by letting the model use only the left context for the next token prediction. In this way, the model keeps the strengths of autoregressive models while exploiting the whole input sequence for better-contextualized output embedding.

Token Detection (TD) was introduced by ELECTRA (Clark et al., 2020), which is an architecture composed of a discriminator and a smaller generator network. First, the generator is trained with MLM and finds suitable replacements for the masked tokens, as in BERT. Then, those candidates are inserted in the original sentence, and the resulting sequence is fed to the discriminator, which classifies whether a token is original or not. TD has the advantage of computing the loss on the whole discriminator output and having a minimal memory footprint. However, the whole system is inefficient because of the presence of the generator, which is still MLM-based.

4 Effective Pre-training Objectives

This section presents our alternative pre-training objectives, which can potentially be applied to a wide range of Transformer-based models.

Random Token Substitution (RTS) Like ELECTRA, RTS trains a model that discriminates between original and substituted tokens. The main difference is that RTS replaces 15% of the tokens with random alternatives, thus avoiding using computational resources to train a separate and expensive network. Besides, unlike MLM, this approach relies on a smaller classification head that is *not* proportional to the vocabulary size, see Figure 1.

[1] Very often the language modelling head parameters are shared with the embedding layer

[2] Gradients have to be computed for every token in the vocabulary because of the final softmax layer

Aggregated probabilities of token misclassification (C-RTS) The random selection applied by RTS may provide too many *easy cases* to perform effective pre-training. Thus, our idea is to use a token probability distribution, inversely proportional to the classification simplicity. More formally, let $(w_0, w_1, \ldots, w_n)$ be an input sequence of tokens. A transformer model m predicts $y = \{0, 1\}^n$, where $y_i = 0$ indicates w_i is original, and $y_i = 1$ that w_i was replaced with some w'_i ($w_i \rightarrow w'_i$). We aim at maximising $P(y_i = 0 \mid w_i \rightarrow w'_i)$, since we want to create substitutions $w_i \rightarrow w'_i$ that are difficult to be detected by m.

$P(y_i = 0 \mid w_i \rightarrow w'_i)$ can be estimated by counting the number of failures/successes in detecting $w_i \rightarrow w'_i$ in previous iterations. While ELECTRA exploits the whole input context to create challenging replacements, our algorithm uses only the prediction history over single tokens. Storing a counter for every pair or token would generate a vast and sparse matrix given the average transformer's vocabulary size. For this reason, we partition tokens into n clusters by measuring the L2 norm between the relative embedding vectors. We train the word embeddings with a *word2vec* model (Mikolov et al., 2013) on the same data used for pre-training, see Appendix F for more details. After that, we use *K-Means* (Lloyd, 1982) to group the tokens into the clusters $C_1, \ldots, C_n$.

We use a matrix $\boldsymbol{F} \in \mathbb{Z}^{n \times n}$, initialized with zeros, to count the difference between the failures and successes of the discriminator. While training, for each pair of tokens $(w_i \in C_a, w'_i \in C_b)$ such that $w_i \rightarrow w'_i$, we decrease $\boldsymbol{F}_{a,b}$ by 1 if $y_i = 0$, otherwise we increment it by 1.

To maximise $P(y_i = 0 \mid w_i \rightarrow w'_i)$ in our approach, we select the pairs (w_i, w'_i) with the highest mistake probability estimated with $\boldsymbol{F}$. We compute the probability of selecting $w_i \in C_a$ and replacing it with $w'_i \in C_b$ as follows:

$$P(w_i \in C_a \rightarrow w'_i \in C_b) = P(w_i)\, P(w'_i|C_b)\, P(C_b|C_a)$$

where we set (i) the probability of selecting a token in the input sequence $P(w_i)$ to 15%; (ii) $P(w'_i|C_b)$ equal to $\frac{1}{|C_b|}$ because it is computed assuming uniform probability of choosing w'_i in C_b; and (iii) we set $P(C_b|C_a)$ as the probability of detecting tokens from cluster C_a when replaced with tokens from C_b. The latter is computed from $\boldsymbol{F}$ as follows: given the candidate token $w_i \in C_a$, we define a multinomial distribution over the target clusters by indexing the a-th row of $\boldsymbol{F}$. To transform the counts in $\boldsymbol{F}_a$ into values interpretable as probabilities, we first apply the *min-max* normalization and then a γ-softmax. γ controls how the probability mass is concentrated or relaxed around the most probable cluster. For our token substitutions, we sample C_b from this multinomial distribution.

We searched for the best n among a reasonable set $\{30, 100, 300, 1000\}$ and the best γ in $\{1, 2, 5, 10\}$. After preliminary experiments, we found that the best combination is $n = 100, \gamma = 2$.

Swapped Language Modeling (SLM) is similar to BERT's MLM, but in this case, tokens are only randomly replaced with others and never with the special MASK. Then, unlike RTS, the model is trained to predict the original value and not discriminate between fakes and originals. SLM uses the same pre-training head as MLM; thus, it is computationally equivalent to it.

5 Datasets

This section contains the descriptions of the datasets we used for pre-training and fine-tuning.

5.1 Pre-training datasets

We used Wikipedia and BookCorpus for pre-training, as in BERT (Devlin et al., 2019). Wikipedia is a large collection of documents containing raw text for a total of about 2,500M words. We cleaned the corpus by removing lists, tables, headers, links and footers, and we considered only the passages. The BookCorpus (Zhu et al., 2015) is composed instead of free novel books, containing approximately 800M words after cleaning. Since the original BookCorpus is not available anymore, we used the version available from the *datasets* library (Lhoest et al., 2021), which may result in slightly different final scores.

5.2 Fine-tuning datasets

GLUE (an acronym for General Language Understanding Evaluation) is a benchmark suite to test models on different NLU tasks (Wang et al., 2019). The collection includes datasets for Question Answering, Natural Language Inference, Question Pairs detection, Entailment Recognition and Language Acceptability. For more details about each task, see Appendix C.

ASNQ (which stands for Answer-Sentence Natural Questions), is a dataset built for Answer Sentence Selection (AS2) (Garg et al., 2019). It was

Dataset	Split	# Questions	# Candidates
ASNQ	Train	57,242	20,377,568
	Dev	2,672	930,062
WikiQA	Train	2,118	20,360
	Dev	122	1,126
	Test	237	2,341
TREC-QA	Train	1,226	53,417
	Dev	69	1,343
	Test	68	1,442

Table 1: Statistics for the three AS2 datasets that we considered (ASNQ, WikiQA, and TrecQA). Notice that we use the "clean" setting for the dev and test splits WikiQA and TrecQA, while for ASNQ we use the original splits proposed in (Garg et al., 2019)

derived from the Natural Questions (NQ) corpus (Kwiatkowski et al., 2019), which was initially designed for Machine Reading. It contains thousands of questions extracted from the Google search engine and candidate sentences retrieved from the top-ranked Wikipedia page. We present the details of the dataset in Table 1.

WikiQA is a small dataset for Answer Sentence Selection built from questions asked to the Microsoft Bing search engine (Yang et al., 2015). Questions have been manually paired with answers taken from Wikipedia articles and labelled as relevant or not. We train on the whole training split, but we validate and test in a "clean" setting: questions having only positive or only negative candidates are removed. More details in Table 1.

TREC-QA is another popular benchmark used for AS2 (Wang et al., 2007). The dataset is created from the TREC-8 to TREC-13 tracks of Question Answering. As in (Garg et al., 2019), we train on the large *train-all* split, but we do validation and testing only on the questions that have at least a positive and a negative answer. Note that *train-all* contains more noise and questions that do not have positive answers. However, it is larger than *train* and allows for a more stable fine-tuning. Training with *train-all* and testing on clean data is the standard setting in TREC-QA. We add more details in Table 1.

6 Experimental Setting

In these experiments, we compare the cost and accuracy of our models with the state-of-the-art methods on GLUE and several AS2 benchmarks. Finally, we summarize results derived from previous work. Then, we provide a description of the training set as well as our pre-training and fine-tuning experiments and results.

6.1 Models

We applied every objective to the same BERT (Devlin et al., 2019) architecture to make a fair comparison. Furthermore, since pre-training time is not proportional to the number of parameters but to the number of computations, we also measure the floating point operations required to do pre-training, as in (Clark et al., 2020). FLOPS indicates the number of mathematical operations performed on the underlying hardware and are independent of the used accelerator (CPU, GPU or TPU) and the model size.

6.2 Pre-training

***Base* models** We pre-trained a model for every alternative objective in the same setting as BERT-*base* (Devlin et al., 2019) to perform a fair comparison. More precisely, we pre-trained models on the English Wikipedia and the BookCorpus dataset for 900K steps with a maximum sequence length of 128 tokens and another 100K steps at 512 with an uncased vocabulary. This saves much pre-training time because the computational complexity of the attention is quadratic in the sequence length.

We use Adam and a triangular learning rate for optimization with a peak value of 10^{-4} and 10K warm-up steps. We use a batch size of 256 examples. More details are given in Appendix A.

Since ELECTRA models require more FLOPS because of the generator, we reduce the number of steps proportionally to the presence of the additional generator, as in (Clark et al., 2020). Thus, we pre-trained ELECTRA for a total of 766K steps, of which 689K with a maximum sequence length of 128 tokens and the remaining 77K at 512.

***Small* models** We pre-trained 4 models using the *small* architecture defined by (Clark et al., 2020) and MLM, SLM, RTS and C-RTS as objectives. We pre-train small models with the same data and hyper-parameters for the *base* models but using a larger batch size of 1024 for 500K steps and always using a reduced maximum sequence length of 128 tokens.

6.3 Fine-tuning

We evaluate the effectiveness of the pre-training objectives described in Section 3 by fine-tuning on four benchmark datasets introduced in Section 5.2.

Model	CoLA matt. corr.	MNLI acc	MRPC acc	QNLI acc	QQP acc	RTE acc	SST-2 acc	STS-B spear	AVG %	Time
BERT-B / MLM+NSP ♣	53.0	82.7	82.8	89.6	88.2	62.8	91.2	80.6	78.9	× 1.00
BERT-B / MLM	53.7	83.6	82.6	89.9	89.1	63.1	92.3	83.6	79.7	× 1.00
BERT-B / RTS	57.3	82.6	81.3	89.3	88.9	66.2	91.7	82.2	79.9	× 0.81
BERT-B / C-RTS	57.3	82.8	81.9	89.4	88.6	60.9	91.0	82.2	79.3	× 0.82
BERT-B / SLM	57.0	83.3	83.1	89.7	88.9	65.0	92.3	83.8	80.4	× 1.00
ELECTRA-B / TD	60.6	83.6	84.1	90.2	89.0	69.1	92.4	85.5	81.8	× 1.05

Table 2: Results on the GLUE test set. Notice that the model with ♣ is the same as Table 9. We took the best models on the development set and submitted them to the GLUE leaderboard. Also, in this case, we don't do best model selection from a pool of candidates, and we do not use ensemble models. Results on the dev. set and significance tests are available in Appendix D.

Model	WikiQA MAP	WikiQA MRR	TREC-QA MAP	TREC-QA MRR	ASNQ → WikiQA MAP	ASNQ → WikiQA MRR	ASNQ → TREC-QA MAP	ASNQ → TREC-QA MRR	ASNQ MAP	ASNQ MRR	Time
BERT-B / MLM+NSP ♣	82.8 (0.9)	84.2 (1.0)	87.2 (0.9)	92.9 (1.0)	87.9 (0.4)	89.3 (0.4)	89.1 (0.4)	93.4 (0.5)	66.8 (0.1)	73.2 (0.2)	× 1.00
BERT-B / MLM	79.9 (1.1)	81.2 (1.2)	86.4 (0.7)	91.5 (0.7)	88.9 (1.0)	90.2 (1.0)	87.6 (0.9)	90.6 (1.1)	65.5 (0.2)	72.2 (0.3)	× 1.00
BERT-B / RTS	78.5 (2.5)	80.1 (2.5)	86.6 (1.5)	91.8 (1.5)	87.9 (0.5)	89.3 (0.5)	88.5 (0.5)	93.4 (0.4)	64.6 (0.1)	71.1 (0.2)	× 0.81
BERT-B / C-RTS	79.0 (1.9)	80.6 (1.6)	87.0 (0.8)	91.8 (1.1)	87.1 (0.6)	88.3 (0.6)	88.7 (0.4)	92.9 (0.7)	64.7 (0.1)	71.5 (0.1)	× 0.82
BERT-B / SLM	80.2 (1.5)	81.7 (1.6)	87.3 (1.3)	92.1 (1.5)	87.7 (0.8)	89.3 (0.7)	87.9 (0.6)	91.0 (0.6)	65.8 (0.3)	72.7 (0.3)	× 1.00
ELECTRA-B / TD	81.8 (1.6)	83.2 (1.6)	86.8 (1.4)	92.2 (1.5)	88.4 (0.4)	89.8 (0.4)	88.9 (0.3)	92.0 (0.5)	64.9 (0.3)	71.7 (0.4)	× 1.05

Table 3: Results on WikiQA and TREC-QA datasets with single-task fine-tuning and after the transfer step on ASNQ. We also report the results on the dev. set of ASNQ optimizing MAP. The NSP loss of the original BERT mainly improves the results without the transfer step. After the transfer on ASNQ, which trains especially the CLS token representation, the difference with our MLM-based BERT is much smaller. We show the standard deviation after 5 runs with different initialization seeds in rounded brackets. We underline results that are significantly different from the BERT-B / MLM baseline after a two-sided T-Test with a significance level equal to 95%.

GLUE We use the same hyper-parameters used in (Liu et al., 2019b). More details are given in Appendix D. We measure Spearman and Matthews's correlation coefficients for STS-B and CoLA respectively, and accuracy for all the other tasks. For every model, we take the best checkpoint on the development set and evaluate it on the GLUE Leaderboard.

ASNQ We train our models on ASNQ using a batch size of 2048 with a learning rate of 1×10^{-5} for 12 epochs with early stopping on the development set. Since most question-answer pairs are shorter than 128 tokens, we use this as the maximum sequence length. We measure the performance using Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR).

WikiQA & ASNQ → WikiQA Following the approach of TANDA (Garg et al., 2019), we fine-tune our models directly on WikiQA and again on WikiQA but after a transfer step on ASNQ. In particular, for each model, we run a hyperparameter search to obtain the best results. We search for the best batch-sizes in $\{32, 64, 128\}$, the best learning rates in $\{2 \times 10^{-6}, 5 \times 10^{-6}, 1 \times 10^{-5}, 2 \times 10^{-5}\}$, and we always use a maximum sequence length of 128 for up to 40 epochs. We repeat each experiment 10 times with different seeds to also report the results' standard deviation. We evaluate the model's performance on the test set using MAP and MRR.

TREC-QA & ASNQ → TREC-QA For these 2 tasks, we adopted the same strategy used for WikiQA and ASNQ → WikiQA. We repeat each experiment 10 times with different seeds; in this case, we evaluate with the same metrics above.

7 Results

We first report the cost of our models in comparison with the state-of-the-art methods in Tables 5 and 6 for *base* and *small* models, respectively. Notice also that RTS and C-RTS use about half the GPU memory of MLM with *base* and $\frac{1}{3}$ with *small* models. Thus, the training time of the former's to complete all the training steps could have been even shorter by increasing the batch size. However, we keep a constant batch size to make a fair comparison. After that, we carry out a comparison in terms of accuracy on GLUE datasets as well as question-answering benchmarks. Additionally, we compare our models with state-of-the-art results published in previous works, pointing out the cost of training them.

Model	WikiQA		TREC-QA		ASNQ		GLUE	Time
	MAP	MRR	MAP	MRR	MAP	MRR	AVG	
BERT-S / MLM	66.9 (1.7)	68.2 (1.6)	80.9 (2.2)	85.8 (2.6)	58.6 (0.3)	66.0 (0.4)	74.1	× 1.00
BERT-S / RTS	71.8 (1.3)	73.5 (1.5)	81.4 (0.5)	87.3 (1.7)	59.8 (0.1)	66.9 (0.2)	75.4	× 0.54
BERT-S / C-RTS	69.4 (1.3)	70.8 (1.0)	81.3 (1.4)	86.5 (0.7)	59.6 (0.2)	67.1 (0.2)	75.7	× 0.54
BERT-S / SLM	69.5 (1.0)	70.8 (1.0)	81.9 (1.1)	87.3 (1.5)	59.2 (0.2)	66.5 (0.3)	75.7	× 1.00

Table 4: Results of RTS, C-RTS and SLM compared with MLM applied to *small* architectures on AS2 benchmarks and GLUE test set. We underline the results that are significantly different from the BERT-S / MLM baseline after a two-sided T-Test with a significance level of 95%. We show the standard deviation after 5 runs with different initialization seeds in rounded brackets.

7.1 *Base* models

GLUE Table 2 shows that all the considered approaches, except for ELECTRA vanilla (MLM+TD), obtain comparable performance on GLUE. In particular, despite a small performance decrease compared with our MLM BERT, the RTS and C-RTS models require about 20% less time to be pre-trained on the same machine. This suggests that using a smaller classification head greatly impacts the training time. The difference could be even broader when considering models using larger vocabularies such as RoBERTa (Liu et al., 2019b) or models with a smaller number of Transformer layers, such as DistilBERT (Sanh et al., 2020) or tiny-BERT (Turc et al., 2019).

It is also worthwhile mentioning that: (i) our BERT-SLM model achieves better performance than MLM BERT (+0.7%), and the original BERT (+1.5%) on average, confirming that removing the MASK token during pre-training is important; (ii) ELECTRA provides superior performance when fine-tuned on small datasets such as CoLA, while in other tasks it shows similar accuracy to SLM and MLM.

For the sake of completeness, we also show the results on the development set of the GLUE datasets in Appendix D.

QA benchmarks We report the performance obtained by the models on a wide range of QA tasks in Table 3. The results on the dev. set of ASNQ are obtained by fine-tuning all models in the same setting. SLM provides the highest results among all models using token-level objectives, scoring even higher than ELECTRA.

On WikiQA and TREC-QA, RTS and C-RTS have mixed results compared to MLM. In some QA tasks, RTS performs slightly better than MLM, while, in others, slightly lower. To show that there is no statistical significant difference in performance between the two models, we do the following test: we split the test set of TREC-QA and WikiQA in 10 parts, and we test the performance of both models on all mini-batches. Finally, we take the difference in performance between the two models on every mini-batch, and we report the mean and standard deviation. We discover that MLM is better than RTS on ASNQ → WikiQA by 1.4 (± 3.8) points while, similarly, RTS outperforms MLM on ASNQ → TREC-QA by 1.1 (± 2.9). The high std. dev. makes the small difference between models insignificant. In contrast, RTS requires 20% less time to be fully pre-trained.

C-RTS shows minor but consistent improvements over RTS in most QA tasks, especially when trained longer (see Appendix H), also featuring a much smaller std. dev. when fine-tuned without the transfer step on ASNQ, which generally reduces the variability of results (Garg et al., 2019). Moreover, similarly to GLUE, the model trained with SLM obtains small but consistent advantages over MLM in MAP and MRR on three benchmarks out of four.

We stress the fact that the only other differences between our BERT models and the original by Devlin et al. (2019) are the BookCorpus pre-training dataset and the additional NSP loss. We use only MLM to provide a fair comparison between token-level objectives. NSP may improve results slightly because it improves the sentence-level representation already while pre-training. This is especially true when models are not trained as long as RoBERTa or ELECTRA, which dropped NSP because it became insignificant for them, and may even hurt performance after many training steps.

7.2 *Small* models

Table 4 shows that RTS and C-RTS on *small* models outperform MLM in most tasks. In addition, RTS also improves over SLM by a wide margin in two of the three considered benchmarks (WikiQA and ASNQ). Besides, thanks to the smaller architecture, RTS greatly impacts the pre-training time. In particular, Table 6 shows that *small* models ex-

Models	BERT-B / MLM	BERT-B / RTS	BERT-B / C-RTS	BERT-B / SLM	ELECTRA-B / TD
LM head complexity	$O(bs \times L \times \lvert V \rvert)$	$O(bs \times L)$	$O(bs \times L)$	$O(bs \times L \times \lvert V \rvert)$	$O(bs \times L \times \lvert V \rvert)$
FLOPS	1.61×10^{19}	1.54×10^{19}	1.54×10^{19}	1.61×10^{19}	1.98×10^{19}
Training time	3d 14h	2d 22h	2d 23h	3d 14h	3d 18h

Table 5: FLOPS used to pre-train each *base* model, language modelling head complexity and real training time on the same machine. bs stands for batch size, L for the input sequence length and $|V|$ is the vocabulary size, equal to about 30K tokens in BERT models. Notice that BERT-RTS and BERT-C-RTS use less memory thanks to the smaller binary classification head (also potentially allowing for larger batch sizes.)

Models	BERT-S / MLM	BERT-S / RTS	BERT-S / C-RTS	BERT-S / SLM
LM head complexity	$O(bs \times L \times \lvert V \rvert)$	$O(bs \times L)$	$O(bs \times L)$	$O(bs \times L \times \lvert V \rvert)$
FLOPS	1.83×10^{18}	1.64×10^{18}	1.64×10^{18}	1.83×10^{18}
Training time	1d 7h	17h	17h	1d 7h

Table 6: FLOPS used to pre-train each *small* model, language modelling head complexity and real training time on the same hardware. See the caption of Table 5 for more details.

ploiting RTS or C-RTS for pre-training consume half the resources used by MLM. We controlled the training of RTS and C-RTS on a small fraction of the pre-training set used for validation. We discovered that the last-epoch F1 scores of RTS and C-RTS in detecting replaced tokens were 96.3 and 94.7, respectively. This confirms that C-RTS is a more challenging task and may be well suited for longer pre-training. RTS, instead, would be more easily solved, thus providing weaker loss signals to the model.

7.3 Models cost

Tables 5 and 6 show the training compute and time required by several models. FLOPS are significant indicators but may reflect different practical performances if the underlying hardware implements special acceleration for some operations. In fact, the training times on our NVIDIA A100 GPU are not perfectly proportional to the model's FLOPS. For example, RTS and C-RTS are much faster to be pre-trained in practice in comparison to MLM and even more than the theoretical FLOPS difference thanks to the smaller memory footprint. Additionally, even by reducing the number of training steps of ELECTRA proportionally to the additional weight of the generator network, as suggested by the authors, ELECTRA still uses more computing than BERT for pre-training.

7.4 Better modelling or just more computing?

We aim to produce models that require less computing budget to achieve performance similar to MLM-based models. Our results show that the performance improvement is logarithmic in the size of the pre-training dataset. At the same time, there is no statistically significant difference between our computationally lighter objectives and more expensive models such as ELECTRA. Indeed, Table 7 shows that top-performing architectures outperform MLM-based models only when trained on much more data. For example, ELECTRA-*Base* uses 21 times more resources than BERT-*Base*, while RoBERTa uses 53 times more. It is impressive the fact that to reach a score of 90.0 on GLUE, a model has to be trained for 2000 times the original BERT.

Model	GLUE	FLOPS
BERT-S / RTS	75.4	$\times$ 0.10
BERT-S / MLM	74.1	$\times$ 0.12
BERT-S / SLM	75.7	$\times$ 0.12
BERT-B / RTS	79.9	$\times$ 0.81
BERT-B / MLM	79.7	$\times$ 1
BERT-B / SLM	80.4	$\times$ 1
BERT-L / MLM+NSP (Devlin et al., 2019)	83.3	$\times$ 12
ELECTRA-B / TD (Clark et al., 2020)	85.7	$\times$ 21
RoBERTa-B / MLM (Liu et al., 2019b)	86.3	$\times$ 53
ELECTRA-L / TD (Clark et al., 2020)	88.6	$\times$ 194
RoBERTa-L / MLM (Liu et al., 2019b)	88.8	$\times$ 200
ALBERT-L / MLM+SOP (Lan et al., 2020)	90.0	$\times$ 1937

Table 7: Large models comparison on the GLUE test set. We report the average accuracy over the different tasks of Table 2, while FLOPS refers to pre-training.

7.5 Does clustering really matter?

We compared our two efficient approaches (RTS and C-RTS) to understand whether selecting more challenging replacement tokens could really improve the model performance on the downstream tasks on a long run. In order to perform this analysis, we pre-trained two *small* models with both ob-

jectives using a different setting. In particular, we increased the sequence length to 512 and trained for 200K steps with a batch size of 1024, thus letting the models (i) see much more tokens and (ii) compute attention scores over long sequences. Then, we evaluated them on five different benchmarks (WikiQA, TREC-QA, ASNQ, MRPC and QNLI). We chose these datasets because they cover a wide range of tasks (AS2, Paraphrasing and NLI) and a wide range of sizes, from the 3.6K examples of MRPC to the 20M of examples in ASNQ. The results shown in Table 8 clearly underline that in every experiment, C-RTS achieves better performance than RTS. For example, it scores between 1 and 3% points over RTS in MAP on the three AS2 datasets. Moreover, it provides more stable results, as it can be seen from the standard deviation across all experiments. We claim that the advantages of C-RTS over RTS derive from a more difficult pre-training objective, which is slower to converge and provides better loss signals in the last training epochs. More details are given in Appendix H.

Dataset	Metric	RTS	C-RTS
WikiQA	MAP	72.2 (0.6)	75.0 (2.4)
TREC-QA	MAP	84.3 (2.4)	85.4 (1.1)
ASNQ	MAP	59.9 (0.2)	60.7 (0.1)
MRPC	Accuracy	81.5 (1.5)	84.2 (0.1)
QNLI	Accuracy	86.9 (0.3)	86.9 (0.1)

Table 8: Performance comparison between RTS and C-RTS on five different benchmarks. The results reported for WikiQA and TREC-QA are on the test set, while for ASNQ, MRPC and QNLI, we report the highest score on the development set. We show the standard deviation after 5 runs with different initialization seeds in rounded brackets.

8 Discussion and Conclusion

In this work, we studied several alternative methods to pre-train Transformer models. Our approaches aim at designing pre-training objectives that (i) match the results of well-known methods using fewer resources or (ii) outperform previous techniques with the same computing budget. This translates into shorter training, lower memory usage, and the possibility of increasing the batch size. Among other benefits, more efficient models can reduce carbon footprint and infrastructure costs. Notice that this advantage is even higher for smaller models since the MLM classification head is not shrunk proportionally to the architecture size (Turc et al., 2019), thus it increases its weight on the computational complexity for smaller models. Moreover, we demonstrate that the MASK token is useless and similar or even better results can be achieved using only token substitution. We reiterate that our objectives could be easily applied to many different transformer models; we chose the BERT setting due to computational resource constraints and easy reproducibility. In Appendix E we provide an overview of negative results. Finally, we show that recent models' performance improvements are mostly driven by longer training phases rather than by more refined architectures.

We evaluated our approaches on several datasets, such as GLUE, WikiQA, TREC-QA and ASNQ. The results show that RTS and C-RTS match the accuracy of MLM in most tasks, requiring a lower amount of computational effort (20% less), while SLM outperforms MLM in most tasks. C-RTS also shows a lower accuracy in detecting replaced tokens, meaning that the task is harder and better suited for longer pre-training sessions.

In addition, we tested our pre-training objectives on smaller transformer architectures. In this scenario, RTS and C-RTS obtained better performances than MLM by requiring half of its time to be pre-trained. For example, RTS outperforms MLM by almost 5 MAP points on WikiQA. This last finding is potentially helpful for training transformer models from scratch with limited resources.

In the future, we plan to explore combinations of our new techniques with efficient architectures such as ALBERT to take advantage of a lighter structure and more effective pre-training objectives.

9 Limitations

Pre-training of large language models is an expensive operation: powerful hardware must be reserved for many days and a lot of energy is consumed. In this paper we explore different pre-training objectives to both save computational power and increase the performance. However, despite the improvements that we propose, the pre-training of language models still requires an incredible amount of resources. For these reasons and our computational constraints, we worked only with *small* and *base* architecture, leaving *large* models as future work.

We trained many language models only on English training data; however, we did not explore the benefits of our alternative objectives if applied in a multilingual setting, but we believe our approaches

could be easily extended to other languages with similar morphology.

Finally, we benchmark our language models on a wide range of tasks, such as Question Similarity, Answer Sentence Selection, Natural Language Inference, etc., but we omitted several other tasks for space limitation.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators.

Nicki Skafte Detlefsen, Jiri Borovec, Justus Schock, Ananya Harsh Jha, Teddy Koker, Luca Di Liello, Daniel Stancl, Changsheng Quan, Maxim Grechkin, and William Falcon. 2022. Torchmetrics - measuring reproducibility in pytorch. *Journal of Open Source Software*, 7(70):4101.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding.

Luca Di Liello, Siddhant Garg, Luca Soldaini, and Alessandro Moschitti. 2022. Pre-training transformer models with sentence-level objectives for answer sentence selection.

William Falcon et al. 2019. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3(6).

Siddhant Garg, Thuy Vu, and Alessandro Moschitti. 2019. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection.

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.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466.

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.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. 2021. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Peter J. Liu, Yu-An Chung, and Jie Ren. 2019a. Summae: Zero-shot abstractive text summarization using length-agnostic auto-encoders.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach.

Stuart P. Lloyd. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26, pages 3111–3119. Curran Associates, Inc.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

A. Radford. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding.

Mengqiu Wang, Noah A. Smith, and Teruko Mitamura. 2007. What is the Jeopardy model? a quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 22–32, Prague, Czech Republic. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yi Yang, Scott Wen-tau Yih, and Chris Meek. 2015. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. ACL - Association for Computational Linguistics.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. Xlnet: Generalized autoregressive pretraining for language understanding.

Rong Zhang, Revanth Gangi Reddy, Md Arafat Sultan, Vittorio Castelli, Anthony Ferritto, Radu Florian, Efsun Sarioglu Kayi, Salim Roukos, Avi Sil, and Todd Ward. 2020. Multi-stage pre-training for low-resource domain adaptation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5461–5468, Online. Association for Computational Linguistics.

Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books.

Appendix

A Pre-training details

We pre-trained on the cleaned versions of the BookCorpus and the English Wikipedia.

For the optimization of both the *small* and the *base* models, we use Adam with a learning rate equal to 10^{-4}, $\epsilon = 10^{-8}$, $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate scheduler is designed to warm up for 10K steps and then decrease linearly. We use a batch size of 256 examples for the *base* models and 1024 for the smalls. Finally, we apply a constant weight decay rate of 0.01, and the dropout probability is set to 0.1.

B Frameworks & Infrastructure

We implemented every model taking advantage of the HuggingFace Transformers library (Wolf et al., 2020), (ii) PyTorch-Lightning for the training framework and the distributed training algorithm (Falcon et al., 2019) and TorchMetrics for classification and AS2 evaluation metrics (Detlefsen et al., 2022).

We performed our pre-training experiments for every model on 8 NVIDIA A100 GPUs with 40GB of memory each, using *fp16* for tensor core acceleration.

C GLUE tasks

The collection includes: (i) two datasets to test performance in paraphrasing capabilities, one composed of questions (QQP) pairs and the other of the sentence pairs (MRPC); (ii) a dataset for question-answer entailment (QNLI) derived from the SQUAD dataset (Rajpurkar et al., 2016); (iii) three datasets for textual entailment (RTE, MNLI and WNLI); (iv) a single dataset (STS-B) to test the model on textual similarity; (v) a dataset (SST-2) to evaluate performance on sentiment analysis and finally (vi) a dataset to check linguistic acceptability (CoLA). As in (Devlin et al., 2019) and (Clark et al., 2020), for stability reasons we avoid testing on WNLI.

We report the results obtained in the development set in Table 9 for both the *base* and the *small* models.

D GLUE hyperparameters and development set results

We used a batch size of 16 with $lr = 1 \times 10^{-5}$ for CoLA and MRPC, a batch size of 16 with a learning rate of 2×10^{-5} for RTE and STS-B, and a batch size of 32 and a learning rate of 1×10^{-5} for MNLI, QNLI, QQP and SST-2. We set the maximum sequence length to 128 for every task. All the GLUE experiments use a triangular learning rate scheduler with 10% of warmup. We train models using half-precision, and optimize them on the development set. Table 9 provides the results on the development set of GLUE for all models.

E Non impactful objectives

E.1 C-RTS sampling within the same cluster

We experimented with a simplification of C-RTS where tokens are always replaced with tokens within the same cluster. We found that it is important to maintain the possibility to sample also from other clusters because the model was able to learn how tokens are clustered after an enough large number of steps.

E.2 Position-based techniques

We experimented with changing the position of tokens, i.e., we masked some positional encoding in an MLM-like approach. The objective was to retrieve the position of the masked tokens in the original sentence. The classification head of this approach is smaller than MLM (slightly larger than RTS, because of the 512 possible positions in BERT), but the results were worse by 3-4% points on GLUE.

We also defined another objective consisting of (i) shuffling input positions of some tokens, and then (ii) predicting their original position. We selected 15% of the tokens, and we permuted them. Although the results were below MLM by 1-2% on the GLUE average, the approach was as much fast as RTS. A combination of position-based objectives with the token-level ones is a possible future direction.

E.3 LM head-on ELECTRA's discriminator

Capitalizing on the good performance reached from SLM, we implemented and evaluated a version of SLM for ELECTRA. Since SLM cannot be directly applied to the ELECTRA discriminator (the predictions are only performed on the output positions corresponding to tampered tokens nullifying the task of detecting fake tokens), we propose an alternative objective called *SLM-all*. In this case, the discriminator has to predict the whole input sentence, estimating which tokens were changed and

Model	CoLA matt. corr.	MNLI acc	MRPC acc	QNLI acc	QQP acc	RTE acc	SST-2 acc	STS-B spear	AVG %
BERT-S/ MLM	42.2 (0.9)	78.8 (0.2)	80.8 (0.9)	85.7 (0.6)	88.6 (0.1)	58.6 (1.3)	89.4 (0.2)	84.4 (0.2)	76.1
BERT-S / RTS	51.2 (1.8)	79.9 (0.2)	81.5 (0.7)	87.9 (0.1)	89.4 (0.1)	61.2 (0.6)	88.4 (0.4)	85.2 (0.2)	78.1
BERT-S / C-RTS	51.6 (0.9)	79.8 (0.1)	81.3 (1.0)	87.0 (0.4)	89.4 (0.1)	61.6 (1.4)	89.5 (0.2)	85.3 (0.3)	78.2
BERT-S / SLM	45.8 (0.5)	79.1 (0.2)	83.3 (0.4)	86.3 (0.3)	89.0 (0.1)	60.8 (1.4)	88.6 (0.4)	86.0 (0.2)	77.4
BERT-B / MLM+NSP ♣	57.6 (1.8)	84.3 (0.4)	82.3 (1.3)	91.0 (0.7)	91.0 (0.2)	68.9 (1.4)	92.6 (0.1)	89.1 (0.3)	82.1
BERT-B / MLM	58.1 (1.0)	83.4 (0.2)	87.5 (0.5)	90.2 (0.3)	90.9 (0.1)	67.4 (1.2)	92.2 (0.3)	87.8 (0.3)	82.2
BERT-B / RTS	58.1 (1.1)	82.7 (0.2)	87.6 (1.0)	89.4 (0.3)	90.9 (0.1)	68.5 (1.4)	91.5 (0.3)	86.6 (0.4)	81.9
BERT-B / C-RTS	57.4 (0.7)	82.0 (0.3)	84.2 (0.4)	89.6 (0.2)	90.6 (0.1)	66.6 (2.4)	91.5 (0.2)	87.0 (0.2)	81.1
BERT-B / SLM	59.6 (1.0)	83.4 (0.2)	87.5 (0.4)	89.9 (0.2)	91.0 (0.1)	69.2 (1.2)	92.1 (0.1)	87.6 (0.3)	82.5
ELECTRA-B / TD	63.4 (1.3)	83.7 (0.2)	87.2 (0.8)	90.4 (0.1)	91.2 (0.1)	74.6 (1.4)	91.4 (0.4)	88.5 (0.2)	83.8

Table 9: Results on GLUE dev. set for both *base* and *small* models (we use the suffixes -B and -S). The symbol ♣ indicates the official BERT-*base* uncased pre-trained model released by (Devlin et al., 2019), which uses an additional NSP loss during the pre-training. We trained every other model in the same setting. For each task, we fine-tune 5 times and take the best model on the development set. We do single task fine-tuning without best model selection or using ensemble models like in (Clark et al., 2020). For each group of our BERT models, we underline results that are statistically different from the MLM baseline model by doing a statistical T-Test with a significance level equal to 95%.

predicting their original values. At the same time, it should only reproduce the input in output for unchanged inputs. As for the other objectives, we evaluated this approach on GLUE, WikiQA, ASNQ and TREC-QA but we obtained generally worst results than some of the other approaches, also with a more expensive training (1.42 times the time required by MLM with *base* models). This gap in the efficiency between SLM-all and the other models is so large because the latter predicts MLM-like token for every input token. Notice that even by reducing the number of training steps of the ELECTRA model by about 25% (to balance the presence of the additional generator with size 1/3), it uses slightly more FLOPS than BERT-MLM. Specifically, it scores 80.02 on the GLUE average score, 78.7, 86.7, 86.7 of MAP in WikiQA, TREC-QA and ASNQ.

F Clustering of tokens embeddings

First, token embeddings have been obtained by training a word2vec (Mikolov et al., 2013) model over the same data used in pre-training. We also tried to use embeddings from an already pre-trained BERT model, but we saw no significant difference in TD accuracy. Therefore, we decided to use word2vec to provide a complete pre-training pipeline and not rely on an already pre-trained checkpoint. We used a context of 2 words on either side of the target tokens and an embedding size of 300. The training algorithm written in PyTorch (Paszke et al., 2019) took less than 10 minutes on the same GPU used for pre-training. Thus this process is not significant concerning the whole pre-training time. We perform clustering of tokens using the K-means algorithm (Lloyd, 1982). We used the CPU implementation of K-means provided by the *scikit-learn* library (Pedregosa et al., 2011), doing 20 random starts. The clustering took approximately 12 minutes on the Intel Xeon Platinum 8275CL in our machine.

G Classification heads

BERT's MLM has to make predictions over the whole vocabulary. For this reason, the last layer of the model should output a probability distribution over the whole vocabulary, which usually contains about 30K tokens. In particular, BERT uses a linear layer of size $H \times |V|$ (H is the hidden size of the model) followed by a softmax to generate values that could be interpreted as probabilities. On the contrary, RTS or C-RTS needs just a binary classification head to predict whether a token is original or fake. This results in a simple linear layer with size $H \times 2$. For this reason, MLM's classification head is usually tens of thousands times larger than RTS's.

H C-RTS vs RTS with longer pre-training.

We demonstrate the superiority of C-RTS over RTS by doing a longer pre-training on a *small* model. We increased the maximum sequence length from 128 to 512 and kept the same batch size of 1024. We train until both models converge and no longer improve, exploiting the same pre-training data used

for the other experiments. After every epoch, we evaluate the checkpoints on various tasks: WikiQA, TREC-QA and MRPC. We avoid very large datasets such as ASNQ because, on *small* models, they underline more the architectural differences than the pre-training technique.

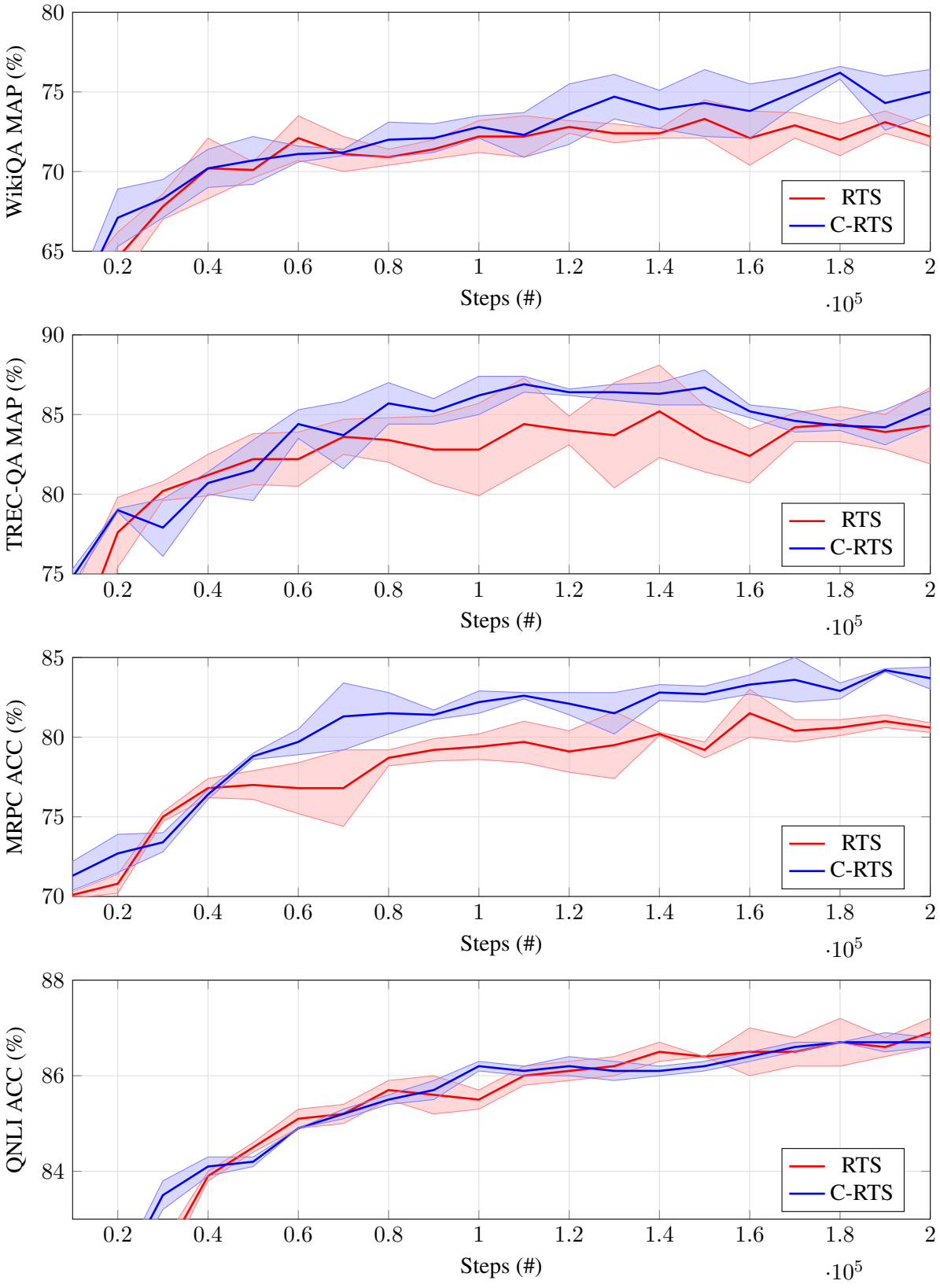

Figure 2: Performance comparison of RTS and C-RTS on WikiQA, TREC-QA, MRPC and QNLI.