Fast-ELECTRA for Efficient Pre-training

Chengyu Dong[†] **Liyuan Liu**[‡] **Hao Cheng**[‡] **Jingbo Shang**[†] **Jianfeng Gao**[‡] **Xiaodong Liu**[‡]
[†]University of California, San Diego [‡]Microsoft Research
{cdong, jshang}@ucsd.edu {lucliu, chehao, jfgao, xiaodl}@microsoft.com

Abstract

ELECTRA pre-trains language models by detecting tokens in a sequence that have been replaced by an auxiliary model. Although ELECTRA offers a significant boost in efficiency, its potential is constrained by the training cost brought by the auxiliary model. Notably, this model, which is jointly trained with the main model, only serves to assist the training of the main model and is discarded post-training. This results in a substantial amount of training cost being expended in vain. To mitigate this issue, we propose Fast-ELECTRA, which leverages an existing language model as the auxiliary model. To construct a learning curriculum for the main model, we smooth its output distribution via temperature scaling following a descending schedule. Our approach rivals the performance of state-of-the-art ELECTRA-style pre-training methods, while significantly eliminating the computation and memory cost brought by the joint training of the auxiliary model. Our method also reduces the sensitivity to hyper-parameters and enhances the pre-training stability.

1 Introduction

ELECTRA (Clark et al., 2020) is a pre-training method that trains the models to predict whether each token is the original token or synthetically generated replacement, in a corrupted input sequence. The token replacement is sampled from a distribution of possible tokens given the context, which is typically the output distribution of a masked language model. Henceforth, we will refer to this masked language model as the auxiliary model [1]. This pre-training task, known as *replaced token detection* (RTD), has shown great advantages in training and data efficiency compared to other pre-training tasks. ELECTRA-style pre-training and its variations have been increasingly popular in advancing natural language understanding capabilities (Meng et al., 2021; Chi et al., 2021; Meng et al., 2022; He et al., 2021; Bajaj et al., 2022).

Despite its effectiveness, one pitfall of ELECTRA that restricts its popularity is the design choices regarding how the auxiliary model is jointly trained with the main language model. This is originally intended to provide a natural curriculum on the RTD task for the main model's learning since the auxiliary model will start off weak and gradually gets better, and thus progressively ramp up the difficulty of the token replacements through pre-training (Clark et al., 2020). However, this design results in a substantial amount of resources (including computation and memory) being wasted, since the auxiliary model will be discarded after each training round. This issue becomes more severe considering that the training cost of the auxiliary model scales with the training cost of the main model, in terms of both the model size and training updates. This issue becomes even more severe considering the practical difficulty of balancing the auxiliary and main model's optimizations (Meng et al., 2022), which often demands multiple training rounds to search for the optimal hyper-parameter.

[1] In previous works (Clark et al., 2020), the auxiliary model is also referred to as the generator while the main model is referred to as the discriminator.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

In this work, we propose a simple ELECTRA-style pre-training alternative that can greatly alleviate this issue. In specific, we employ an existing language model as the auxiliary model, which can either be retrieved from a public repository or from a previous training round. However, directly pre-training with this existing language model impairs the performance, potentially because the auxiliary model generates token replacements that are too difficult. To reduce the difficulty of the token replacements, we smooth the output distribution of the auxiliary model by temperature scaling and gradually decrease the temperature through pre-training following a pre-defined descending schedule. Such a strategy effectively constructs a curriculum for the main model's learning without the need to jointly train the auxiliary model. Our method, referred as Fast-ELECTRA, achieves comparable performance to state-of-the-art ELECTRA-style pre-training methods in various pre-training settings, while being more efficient on top of the already competitive efficiency improvement offered by ELECTRA.

2 Preliminaries

Masked Language Modeling. The masked language modeling (MLM) (Devlin et al., 2019) task used in BERT (Devlin et al., 2019) trains the language model to predict randomly masked tokens in a sequence. Specifically, given an input sequence $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]$, MLM generates a *masked sequence* $\boldsymbol{x}_{\text{masked}}$ by randomly selecting a few tokens at positions $\boldsymbol{m} = [m_1, m_2, \cdots, m_K]$ replace them with `[mask]`. The model is then trained to predict the original tokens at the masked positions. The training objective is $L_{\text{MLM}}(\theta) = \mathbb{E}_{\boldsymbol{x}} \sum_{i \in \boldsymbol{m}} - \log p_\theta(x_i | i, \boldsymbol{x}_{\text{masked}})$, where $p_\theta(\cdot | i, \boldsymbol{x}_{\text{masked}})$ is the output distribution of the model θ at position i, conditioned on the masked sequence $\boldsymbol{x}_{\text{masked}}$.

ELECTRA-style Pre-training. Unlike MLM, ELECTRA trains the language model to detect replaced tokens in a sequence. Specifically, given an input sequence $\boldsymbol{x}$, ELECTRA generates a *corrupted sequence* $\boldsymbol{x}_{\text{corrupted}}$ by randomly selecting a few positions $\boldsymbol{m}$ and replace the token x_i at each position $i \in \boldsymbol{m}$ with a corresponding token $\hat{x}_i$ that is likely semantically similar but not necessarily the same. We will refer to $\hat{x}$ as the *replaced token*, which is sampled from a probability distribution over the entire vocabulary. The language model is then trained to predict whether each token in $\boldsymbol{x}_{\text{corrupt}}$ is the original token or a replacement, namely $L_{\text{RTD}}(\theta) = \mathbb{E}_{\boldsymbol{x}} \sum_{i=1}^{n} -1(\hat{x}_i \neq x_i) \log p_\theta(i, \boldsymbol{x}_{\text{corrupt}}) - 1(\hat{x}_i = x_i) \log(1 - p_\theta(i, \boldsymbol{x}_{\text{corrupt}}))$, where $p_\theta(i, \boldsymbol{x}_{\text{corrupt}})$ is the probability of replacement predicted by the model at position i, and $1(\cdot)$ is the indicator function.

ELECTRA-style pre-training has greatly improved the training efficiency compared to MLM and has dominated the state-the-of-arts on natural language understanding benchmarks (He et al., 2021; Meng et al., 2022; Bajaj et al., 2022). For example, ELECTRA can rival the downstream performance of RoBERTa (Liu et al., 2019b), a competitive MLM-style pre-training method, with only 25% of the computation cost (Clark et al., 2020).

Auxiliary Model and Joint-training. The pivot of the RTD task in ELECTRA-style pre-training is the probability distribution which the replaced tokens are sampled from, which is typically determined by an auxiliary masked language model. Specifically, to generate the corrupted sequence $\boldsymbol{x}_{\text{corrupted}}$ mentioned above, ELECTRA first replaces all the tokens at positions $\boldsymbol{m}$ with `[mask]` and obtain a masked sequence $\boldsymbol{x}_{\text{masked}}$. The probability distribution of the replaced token at each position is then simply the corresponding output distribution of the auxiliary model evaluated on $\boldsymbol{x}_{\text{masked}}$, namely $\hat{x}_i \sim p_{\text{aux}}(\cdot | i, \boldsymbol{x}_{\text{masked}})$ for each $i \in \boldsymbol{m}$.

In the original ELECTRA design, the auxiliary model is jointly trained with the main model, which is shown to be necessary for the effectiveness of the pre-training (Clark et al., 2020). The overall training objective is defined as $\min_{\theta, \theta_{\text{aux}}} L_{\text{MLM}}(\theta_{\text{aux}}) + \lambda L_{\text{RTD}}(\theta)$, where θ_{aux} refers to the parameters of the auxiliary model, and λ is a hyper-parameter that balances the optimizations of the auxiliary model and the main model. The intuition here is that joint training can provide a natural curriculum on the RTD task for the main model's learning since the difficulty of the replaced tokens will gradually increase, as the auxiliary model improves throughout pre-training.

3 Computation Overhead Reduction of Auxiliary Model

Computing and Memory Cost of the Auxiliary Model. Despite that joint training of the auxiliary model is effective, it results in a significant amount of computation resources being wasted since the auxiliary model will be discarded once the training is finished. Our estimations show that, in each training trial, in each training update, and for each input batch, the auxiliary model expends about 67% of the computation cost (about 4.0e11 FLOPs) of the main model when pre-training a BERT-base equivalent model. The overall computation cost scales dramatically as one pre-trains with larger batch size, for more updates, and more hyper-parameter searching rounds. The auxiliary model also consumes a significant amount of memory during training, which is about 30% of the memory cost (about 7 GB) of the main model when pre-training a BERT-base equivalent model. The excessive memory cost of ELECTRA could in turn induce more computation cost than necessary because only smaller batch sizes can fit into the memory.

Fast-ELECTRA. We propose a simple alternative to significantly reduce the computation and memory cost of the auxiliary model inspired by simulated annealing (Kirkpatrick et al., 1983). Specifically, we employ an existing language model as the auxiliary model, which can either be retrieved from a public repository or from a previous training experiment. To reduce the difficulty of the RTD task, we leverage temperature scaling to smooth the output distribution of this existing model. The replaced tokens in the RTD task are then sampled from its smoothed output distribution,

$$\hat{x}_i \sim \text{Softmax}\left(\frac{\log p_{\text{aux}}(\cdot|i, \boldsymbol{x}_{\text{masked}})}{T}\right), \tag{1}$$

To create a learning curriculum for the main model similar to the effect of joint training, we simply schedule the temperature T by an exponential decay function during pre-training, namely

$$T = 1 + (T_0 - 1) \cdot \exp(-u/\tau). \tag{2}$$

Here u denotes the fraction of the training updates, while T_0 and τ are two hyper-parameters that control the initial temperature and decay rate respectively. Since now the auxiliary model is not jointly trained, the training objective is simply $\min_\theta L_{\text{RTD}}(\theta)$.

4 Experiments

The main advantage of Fast-ELECTRA is training efficiency as our auxiliary model is only used for inference during pre-training. The computation cost of the auxiliary model is now only $1/3$ of the original, while the memory cost of the auxiliary model is now only about $1/30$ of the original. With offline preprocessing, we can further reduce both the computation and memory cost of the auxiliary model during pre-training to 0. We will mainly present these experiment results in this section.

Our experiments follow the standard natural language understanding setup (Devlin et al., 2019; Meng et al., 2021; Bajaj et al., 2022), which includes both *Base* and *Large* settings, correspond to BERT$_{\text{base}}$ and BERT$_{\text{large}}$ architecture respectively (See more in Section C.1). While being more efficient, Fast-ELECTRA matches previous state-of-the-arts with jointly-trained generator on both the Base and Large setting (see Section C.2). Our design also improves ELECTRA's robustness to the hyper-parameter settings (see Section C.3) and its training stability (see Section C.4).

Computation Cost. The computation cost of ELECTRA is contributed by both the main model and the auxiliary model. We assume the backward propagation has approximately twice the computation cost of the forward propagation following (Kaplan et al., 2020). Therefore, the computation cost of the auxiliary model is reduced by $2/3$ since it is only used for inference in Fast-ELECTRA.

We estimate the computation cost more accurately by calculating training FLOPs for each input batch in each training update (*i.e.*, one forward plus one backward propagation). We follow the formula introduced by (Hoffmann et al., 2022) to calculate the FLOPs for both the main model and the auxiliary model. As shown in Table 1, our method can reduce the overall computation cost by about 20-25% for both base and large models.

Memory cost. The memory cost of ELECTRA is contributed by both the main model and the auxiliary model. Considering standard training setup of language models such as the Adam optimizer (Kingma & Ba, 2015a) and mixed-precision training (Micikevicius et al., 2017), the memory cost of each trainable parameter is contributed by its weight, its gradient, and its corresponding state buffers in the optimizer, which requires 20 bytes in total (Smith et al., 2022). In contrast, for a parameter that is only used in inference, the memory cost consists of only its weight, namely 2 bytes. Therefore, the memory cost of the auxiliary model is significantly reduced since it is only used for inference in Fast-ELECTRA.

Besides the model parameters, the intermediate activations of the computation graph stored for backward propagation consume significant memory as well. In typical implementations, one can

Table 1: Computation cost per batch per update and memory consumption of ELECTRA. Note that for the original ELECTRA, we exclude the embedding layer from the memory calculation of the auxiliary model since it is shared between the auxiliary model and the main model. We also report the computation and memory cost on realistic computation infrastructures in Appendix E.1.

Model	Method	Computation (GFLOPs)			Memory (GB)		
		Main	Auxiliary	Total	Main	Auxiliary	Total
Base	Original	591.9	398.6	990.5	23.0	7.0	30.0
	Fast-ELECTRA	591.9	132.9	724.8	23.0	0.25	23.3
	Ratio	1.0	0.33	0.73	1.0	0.04	0.77
Large	Original	1407.7	653.9	2061.6	60.2	14.4	74.6
	Fast-ELECTRA	1407.7	218.0	1625.6	60.2	0.42	60.6
	Ratio	1.0	0.33	0.79	1.0	0.03	0.81

employ checkpointing (Gruslys et al., 2016; Chen et al., 2016) to trade computation for memory and reduce the memory footprint within each encoding layer. Nevertheless, the activations after each encoding layer still need to be stored (Smith et al., 2022). Therefore, the activation memory scales with the number of layers, which means the auxiliary model always consumes about $1/4$-$1/3$ additional memory on top of the main model. The activation memory also scales with the sequence length, number of hidden dimensions, and batch sizes. In contrast, the auxiliary model in Fast-ELECTRA consumes 0 intermediate activation memory since it is used only for inference.

As shown in Table 1, our method can reduce the memory cost[2] of the auxiliary model by about 97% and the overall memory cost by more than 20% for both base and large models.

Offline preprocessing for 0 auxiliary computation and memory cost. Last but not least, Fast-ELECTRA also makes the offline preprocessing technique feasible for ELECTRA-style pre-training, which can further reduce both the computation and memory cost of the auxiliary model to 0 during pre-training. In specific, one can generate and dump the training data of the RTD task (*i.e.*, the corrupted input sequence and the binary target indicating whether each token is replaced or not) at each epoch before pre-training, since the only varying parameter in our auxiliary model is the temperature and it can be determined by the epoch number. These dumped training data can be reused in subsequent pre-training rounds, for example, for hyper-parameter search or continual pre-training. This strategy would be particularly preferred for large-scale pre-training.

It is worth noting that Fast-ELECTRA equipped with offline preprocessing can be even more computation-efficient than MLM with offline preprocessing (known as static masking (Devlin et al., 2019; Liu et al., 2019b)) since the training targets are binary instead of scale with the vocabulary.

We conducted ablation studies on the design of Fast-ELECTRA. Our experiments show that it is possible to sample replaced tokens in ELECTRA from pre-defined distributions that require no auxiliary model, which would potentially improve efficiency further, albeit they are inferior to auxiliary-model-based ones in terms of performance (See Section D.1). Our additional experiments also show that sampling replaced tokens from a fixed pre-trained model significantly hurts the performance, demonstrating the necessity of the learning curriculum(See Section D.2). Finally, our experiments also show that it is possible to design the learning curriculum through alternative ways such as Dropout (See Section E.2).

5 Conclusion

In this work, we focus on the training cost of the auxiliary model in ELECTRA-style pre-training and propose a simple method that employs existing language models and annealed temperature scaling to greatly alleviate the issue. Our method achieves comparable performance to state-of-the-art while being more efficient and robust. In general, our approach empowers ELECTRA-style pre-training with more flexibility, opening up potential applications in continual learning, transfer learning, and knowledge distillation for language models.

[2] For the activation memory, we account for the entire batch size and ignore gradient accumulation here as it depends on the specific GPU memory size.

References

Arora, S., Cohen, N., Golowich, N., and Hu, W. A convergence analysis of gradient descent for deep linear neural networks. *ArXiv*, abs/1810.02281, 2018.

Bajaj, P., Xiong, C., Ke, G., Liu, X., He, D., Tiwary, S., Liu, T.-Y., Bennett, P., Song, X., and Gao, J. Metro: Efficient denoising pretraining of large scale autoencoding language models with model generated signals. *ArXiv*, abs/2204.06644, 2022.

Bao, H., Dong, L., Wei, F., Wang, W., Yang, N., Liu, X., Wang, Y., Piao, S., Gao, J., Zhou, M., and Hon, H.-W. Unilmv2: Pseudo-masked language models for unified language model pre-training. *ArXiv*, abs/2002.12804, 2020.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *ArXiv*, abs/1604.06174, 2016.

Chi, Z., Huang, S., Dong, L., Ma, S., Singhal, S., Bajaj, P., Song, X., and Wei, F. Xlm-e: Cross-lingual language model pre-training via electra. In *Annual Meeting of the Association for Computational Linguistics*, 2021.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. Electra: Pre-training text encoders as discriminators rather than generators. *ArXiv*, abs/2003.10555, 2020.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. Memory-efficient backpropagation through time. In *NIPS*, 2016.

Hao, Y., Dong, L., Bao, H., Xu, K., and Wei, F. Learning to sample replacements for electra pre-training. In *FINDINGS*, 2021.

He, P., Liu, X., Gao, J., and Chen, W. Deberta: Decoding-enhanced bert with disentangled attention. *ArXiv*, abs/2006.03654, 2020.

He, P., Gao, J., and Chen, W. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *ArXiv*, abs/2111.09543, 2021.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models. *ArXiv*, abs/2203.15556, 2022.

Kaplan, J., McCandlish, S., Henighan, T. J., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *ArXiv*, abs/2001.08361, 2020.

Ke, G., He, D., and Liu, T.-Y. Rethinking positional encoding in language pre-training. *ArXiv*, abs/2006.15595, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015a.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015b.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. Optimization by simulated annealing. *Science*, 220:671 – 680, 1983.

Liu, L., Liu, X., Gao, J., Chen, W., and Han, J. Understanding the difficulty of training transformers. *ArXiv*, abs/2004.08249, 2020a.

Liu, L., Liu, J., and Han, J. Multi-head or single-head? an empirical comparison for transformer training. *ArXiv*, abs/2106.09650, 2021.

Liu, X., He, P., Chen, W., and Gao, J. Multi-task deep neural networks for natural language understanding. In *Annual Meeting of the Association for Computational Linguistics*, 2019a.

Liu, X., Wang, Y., Ji, J., Cheng, H., Zhu, X., Awa, E., He, P., Chen, W., Poon, H., Cao, G., et al. The microsoft toolkit of multi-task deep neural networks for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 118–126, 2020b.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019b.

Meng, Y., Xiong, C., Bajaj, P., Tiwary, S., Bennett, P., Han, J., and Song, X. Coco-lm: Correcting and contrasting text sequences for language model pretraining. *ArXiv*, abs/2102.08473, 2021.

Meng, Y., Xiong, C., Bajaj, P., Tiwary, S., Bennett, P., Han, J., and Song, X. Pretraining text encoders with adversarial mixture of training signal generators. *ArXiv*, abs/2204.03243, 2022.

Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., García, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed precision training. *ArXiv*, abs/1710.03740, 2017.

Newman, M. E. J. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46:323 – 351, 2004.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *North American Chapter of the Association for Computational Linguistics*, 2019.

Raffel, C., Shazeer, N. M., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683, 2019.

Rasley, J., Rajbhandari, S., Ruwase, O., and He, Y. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V. A., Zhang, E., Child, R., Aminabadi, R. Y., Bernauer, J., Song, X., Shoeybi, M., He, Y., Houston, M., Tiwary, S., and Catanzaro, B. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *ArXiv*, abs/2201.11990, 2022.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *BlackboxNLP@EMNLP*, 2018.

Xu, Z., Gong, L., Ke, G., He, D., Zheng, S., Wang, L., Bian, J., and Liu, T.-Y. Mc-bert: Efficient language pre-training via a meta controller. *ArXiv*, abs/2006.05744, 2020.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J. G., Salakhutdinov, R., and Le, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. In *Neural Information Processing Systems*, 2019.

Zhang, Z., Zhao, H., Utiyama, M., and Sumita, E. Language model pre-training on true negatives. *ArXiv*, abs/2212.00460, 2022.

Zhu, Y., Kiros, R., Zemel, R. S., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 19–27, 2015.

A Related Work

Variations of ELECTRA-style Pre-training. Here we briefly summarize the variations of the ELECTRA-style pre-training in the literature. Xu et al. (2020) pre-trains the model to predict the original token from a small candidate set, instead of predicting a binary target. Meng et al. (2021) introduces two additional training objectives including the prediction of the original token and alignment between corrupted sequences from the same source. Hao et al. (2021) learns to sample more difficult replace tokens. Meng et al. (2022) automatically constructs a difficult learning signal by an adversarial mixture of multiple auxiliary models. He et al. (2021) argues that embedding sharing between the main model and the auxiliary model may hurt pre-training and proposes a stop gradient operation during back-propagation. Bajaj et al. (2022) conducts a comprehensive ablation study on ELECTRA and highlights several important improvements such as large vocabulary size and relative position embedding, and successfully scales ELECTRA-style pre-training up to billions of parameters. Zhang et al. (2022) observes the existence of "false negative" replaced tokens, namely those that are not exactly the same but are synonyms to the original ones, and proposes to correct them by synonym look-up and token similarity regularization.

B Hyper-parameter Settings

We follow the standard practice in previous works to set the hyper-parameters (Devlin et al., 2019; Meng et al., 2021; Bajaj et al., 2022). For MLM pre-training of the generator, we fix the mask ratio as 15%. When sampling sequences for pre-training, we respect document boundaries and avoid concatenating texts from different documents. We did not mask special tokens follow the standard BERT practice. We conduct pre-training on NVIDIA Tesla V100 with 32GB memory and fine-tuning on NVIDIA Tesla P100 with 16GB memory. Table 3 lists the detailed hyper-parameter settings for pre-training. Table 4 lists the detailed hyper-parameters used for fine-tuning.

Table 2: Configuration of model architectures. Note that the auxiliary model has the same configuration in each encoding layer as the main model, albeit having fewer layers. As a reference for the calculation of memory cost, we list the embedding layer separately since it is shared between the main model and the auxiliary model in the original ELECTRA design.

Model	Depth (Main)	Depth (Aux)	Hidden Size	FFN Width	Attention Heads	# Params (Main)	# Params (Aux)	# Params (Embed)
Base	12	4	768	3072	12	184 M	127M	98M
Large	24	6	1024	4096	16	434 M	208M	131M

Table 3: Hyperparameter settings for pre-training.

Hyperparameters	**Base**	**Large**
Max Steps	125K	125K
Optimizer	Adam	Adam
Peak Learning Rate (Fast-ELECTRA)	1×10^{-3}	1×10^{-3}
Peak Learning Rate (METRO$_{\text{ReImp}}$)	5×10^{-4}	5×10^{-4}
Loss Weight (METRO$_{\text{ReImp}}$)	70	50
Batch Size	2048	2048
Warm-Up Steps	10K	10K
Sequence Length	512	512
Vocabulary Size	128K	128K
Relative Position Encoding Buckets	32	128
Relative Position Encoding Max Distance	128	256
Adam ϵ	1e$-$6	1e$-$6
Adam (β_1, β_2)	$(0.9, 0.98)$	$(0.9, 0.98)$
Clip Norm	2.0	2.0
Dropout	0.1	0.1
Weight Decay	0.01	0.01

Table 4: Hyperparameter search space in fine-tuning.

Hyperparameters	Base	Large
Sequence Length	256	256
Optimizer	AdaMax	AdaMax
Peak Learning Rate	{5e-5,1e-4, 3e-4}	{5e-5,1e-4, 3e-4}
Max Epochs	{2,3,5,10}	{2,3,5,10}
Batch size	{16, 32, 64}	{16, 32, 64}
Learning rate decay	Linear	Linear
Weight Decay	{0, 0.01}	{0, 0.01}
Warm-up Proportion	{6 %, 10 %}	{6 %, 10 %}
Adam ϵ	1e-6	1e-6
Adam (β_1, β_2)	$(0.9, 0.98)$	$(0.9, 0.98)$
Gradient Clipping	1.0	1.0
Dropout	0.1	0.1

C Experiments

C.1 Experiment Setup

Pre-training Setup. We conduct experiments with two standard settings, *Base* and *Large*, following previous works (Devlin et al., 2019; Meng et al., 2021; Bajaj et al., 2022). Specifically, we employ Wikipedia and BookCorpus (Zhu et al., 2015) (16 GB of texts, 256M samples) for pre-training with a sequence length of 512. We use a cased sentence piece BPE vocabulary of 128K tokens following (He et al., 2020), since larger vocabulary size improves LLMs without significant additional training and inference cost (Bao et al., 2020). We conduct pre-training for 125K updates with a batch size of 2048. For Fast-ELECTRA, the auxiliary model is pre-trained following a standard MLM style with a learning rate of 5e-4.

Model Architecture. Our main model (discriminator) in the Base setting follows the BERT$_{\text{base}}$ architecture (Devlin et al., 2019), namely a 12-layer transformer with 768 hidden dimensions plus T5 relative position encoding (Raffel et al., 2019) with 32 bins. We employ Admin (Liu et al., 2020a, 2021) for model initialization to stabilize the training. Our main model in the Large setting follows BERT$_{\text{Large}}$, namely a 24-layer transformer with 1024 hidden dimensions and 128 relative position encoding bins. We follow previous works (Clark et al., 2020; Bajaj et al., 2022) to set the size of the auxiliary model (generator), namely 4 layers for the Base setting and 6 layers for the large setting. More details of the model configuration are listed in Table 2.

Downstream evaluation setup. We conduct evaluation on downstream tasks following the setup in previous works (Meng et al., 2021; Bajaj et al., 2022). Specifically, we evaluate on GLUE (Wang et al., 2018) language understanding benchmark with a single-task, single-model fine-tuning setting following previous works. We report Spearman correlation on STS-B, Matthews correlation on CoLA, and accuracy on the rest of the datasets. We follow the training hyperparameters suggested by Liu et al. (2019a, 2020b), such as the use of an AdaMax optimizer (Kingma & Ba, 2015b). Detailed hyperparameter settings can be found in Appendix B.

Baselines. We compare our method with various baselines with an experiment setup same as ours, in terms of dataset, model size, and computation cost. We also incorporate baselines with similar experiment setup to allow more comparisons. For example, in the Large setting, we report baselines that pre-train for 1M updates but with a bach size of 256, which aligns our setup in terms of the total number of processed tokens. We obtain results of these baselines from their papers and follow-up works, whichever are higher. We also reimplement METRO as our baselines. Both the re-implemented METRO (Bajaj et al., 2022) and our method are implemented within the same codebase, which is built on top of FAIRSEQ (Ott et al., 2019), a popular open-sourced package.

Hyper-parameter Settings. We follow previous works (Clark et al., 2020; Bajaj et al., 2022) to select the generator size, namely 4 layers for the Base setting and 6 layers for the Large setting. For our re-implemented METRO, we set the loss weight as 70 and 50 for the Base and Large setting respectively while the learning rate as 5e-4 for both settings, which produces the best results in our experiments. For Fast-ELECTRA, we set the initial temperature as 2, the decay rate as 0.1, and the

Table 5: Results on GLUE development set. “-” indicates that no public reports are available. “†” indicates the model is pre-trained for 1M updates with batch size of 256. “‡” indicates the model is pre-trained for 100K updates with batch size of 8K.

Model	**MNLI-(m/mm)** **(Acc.)**	**QQP** **(Acc.)**	**QNLI** **(Acc.)**	**SST-2** **(Acc.)**	**CoLA** **(Mat. Corr.)**	**RTE** **(Acc.)**	**MRPC** **(Acc.)**	**STS-B** **(Spear. Corr.)**	**Average** **Score**
Base Setting									
BERT (Devlin et al., 2019)	84.5/ -	91.3	91.7	93.2	58.9	68.6	87.3	89.5	83.1
RoBERTa (Liu et al., 2019b)	85.8/85.5	91.3	92.0	93.7	60.1	68.2	87.3	88.5	83.3
XLNet (Yang et al., 2019)	85.8/85.4	-	-	92.7	-	-	-	-	-
DeBERTa (He et al., 2020)	86.3/86.2	-	-	-	-	-	-	-	-
TUPE (Ke et al., 2020)	86.2/86.2	91.3	92.2	93.3	63.6	73.6	89.9	89.2	84.9
ELECTRA (Clark et al., 2020)	86.9/86.7	91.9	92.6	93.6	66.2	75.1	88.2	89.7	85.5
MC-BERT (Xu et al., 2020)	85.7/85.2	89.7	91.3	92.3	62.1	75.0	86.0	88.0	83.7
COCO-LM (Meng et al., 2021)	88.5/88.3	92.0	93.1	93.2	63.9	84.8	**91.4**	90.3	87.2
AMOS (Meng et al., 2022)	88.9/88.7	**92.3**	93.6	94.2	**70.7**	**86.6**	90.9	**91.6**	88.6
DeBERTaV3 (He et al., 2021)	**89.3/89.0**	-	-	-	-	-	-	-	-
METRO (Bajaj et al., 2022)	89.0/88.8	92.2	93.4	**95.0**	70.6	86.5	91.2	91.2	88.6
METRO$_{\text{ReImp}}$	89.0/88.9	92.0	93.4	94.4	70.1	86.3	**91.4**	91.2	88.5
Fast-ELECTRA	89.4/88.8	92.1	**93.8**	94.5	**71.4**	85.6	**91.4**	**91.6**	**88.7**
Large Setting									
BERT[†]	86.6/ -	-	-	-	-	-	-	-	-
RoBERTa[‡]	89.0/ -	91.9	93.9	**95.3**	66.3	84.5	90.2	91.6	87.8
XLNet[†]	88.4/ -	91.8	93.9	94.4	65.2	81.2	90.0	91.1	87.0
TUPE[†]	88.2/88.2	91.7	93.6	95.0	67.5	81.7	90.1	90.7	87.3
METRO$_{\text{ReImp}}$	89.9/90.2	**92.5**	**94.5**	94.3	69.7	**88.8**	**91.9**	91.6	89.2
Fast-ELECTRA	**90.1/90.2**	92.4	**94.5**	95.1	**72.1**	87.4	90.7	**91.9**	**89.3**

learning rate as 1e-3 for both the Base and Large settings. Detailed hyper-parameter settings are elaborated in Appendix B.

C.2 Downstream Performance

Table 5 in the appendix lists the downstream evaluation results under the Base and Large setting. Fast-ELECTRA matches previous state-of-the-arts with jointly-trained generator, in terms of the overall GLUE score and results on reliable datasets such as MNLI.

C.3 Robustness to the Hyper-parameter Settings

In this section, we investigate the robustness of our method to the hyper-parameter settings. We are mostly interested in hyper-parameters that control the learning curriculum, namely the difficulty of the RTD task throughout pre-training, since it is the key difference between Fast-ELECTRA and the original ELECTRA and is also important to the pre-training's effectiveness.

In practice, we would strongly prefer an ELECTRA-style pre-training method that is friendly to the learning curriculum tuning. This is because it is challenging to control the learning curriculum since a positive curriculum in the short term is not necessarily beneficial to the entire training course. Searching for the optimal curriculum often requires training till the end. Controlling the learning curriculum is particularly challenging for pre-training because the effectiveness of a curriculum can only be revealed by the performance on representative downstream tasks, which requires time-consuming fine-tuning. There is no reliable and instant signal in pre-training that can indicate the effect of a curriculum.

Size of the auxiliary model. The size of the auxiliary model can directly affect the learning curriculum. A large auxiliary model can converge faster (Arora et al., 2018), thus creating a more difficult RTD task for the main model. Previous works have observed that an auxiliary model that is too large or too small can both damage the effectiveness of the pre-training (Clark et al., 2020). However, tuning the size of the auxiliary model in practice can be overly problematic, as the model architecture changes in each training attempt, which means the experiment or hardware setup very likely needs to be re-configured to maintain the training efficiency.

We show that compared to the original ELECTRA, Fast-ELECTRA is more robust to the size of the auxiliary model, thus being more friendly to use in practice. We experiment on the original ELECTRA and Fast-ELECTRA with multiple auxiliary model depths, while for other hyper-parameters (*e.g.*, loss weight λ in the original ELECTRA and decay rate τ in Fast-ELECTRA), we select the ones that

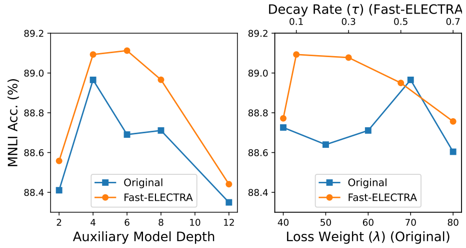

Figure 1: Downstream task performance (MNLI accuracy, Avg m/mm) versus the depth of the auxiliary model (Left) and the hyper-parameters (Right) used in the "Original" ELECTRA or "Fast-ELECTRA".

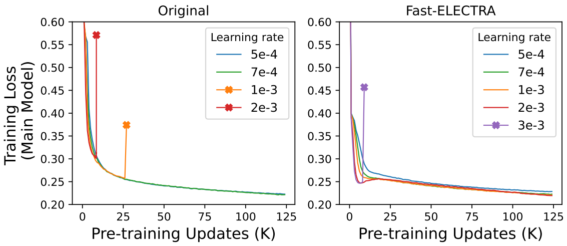

Figure 2: Training loss curves of the main model when pre-training with the original ELECTRA (*Left*) and Fast-ELECTRA (*Right*). "×" indicates the divergence of the training.

produce the best performance for each depth respectively. As shown in Figure 1 left, Fast-ELECTRA can achieve more predictable performances as the auxiliary model depth varies.

Curriculum Schedule. We also experiment on hyper-parameters provided by each method that can directly control the learning curriculum when the auxiliary model is determined. In the original ELECTRA, it is mainly the loss weight λ that controls the learning curriculum, since it balances the optimizations of the auxiliary model and the main model. We neglect the learning rate here since it affects the auxiliary model and the main model simultaneously. In Fast-ELECTRA, the decay rate τ is used to control the learning curriculum. We neglect the initial temperature here since the default value of 2 works well across different auxiliary model and main model settings.

To fairly compare the sensitivities of the performance to λ and τ given their different scales, we sweep each hyper-parameter around two of its best values under different settings. For example, we modulate λ in [40, 50, 60, 70, 80], since 70 and 50 are the best values we found for the Base and Large settings respectively. We modulate τ in [0.05, 0.1, 0.3, 0.5, 0.7] since 0.1 and 0.5 are the best values we found for a 4-layer auxiliary model and a 12-layer auxiliary model respectively.

Figure 1 right shows the downstream performances obtained by experimenting with these hyper-parameter values. It can be seen that the performance changes abruptly when λ varies in the original ELECTRA. In contrast, Fast-ELECTRA produces a smoother performance curve as τ varies.

C.4 Training Stability

In this section, we study the training stability, which is known to be a major bottleneck in training large language models (Liu et al., 2020a). Upon scaling up the model size, one may need to reduce the learning rate, scale down the variance of the weight initialization, apply heavy gradient clipping, or re-configure the model architecture (Bajaj et al., 2022; Smith et al., 2022). However, these remedies often sacrifice the efficiency and/or effectiveness of the pre-training (Bajaj et al., 2022).

Although we cannot afford large-scale experiments, we make an initial attempt to inspect the training stability with a standard model size (BERT-base equivalent). In specific, we conduct pre-training with excessively large learning rates. Figure 2 shows that the original ELECTRA quickly diverges as the learning rate increases, while pre-training with Fast-ELECTRA can remain stable with a learning rate as large as 2e-3, almost triple the maximum value allowed by the original method.

D Ablation Studies

D.1 ELECTRA-style Pre-training Free of Auxiliary Models

Here, we attempt to construct an RTD task without an auxiliary model. As mentioned in Section 2, the pivot of RTD task is the probability distribution which the replaced tokens are sampled from, which we will refer to as the replaced token distribution. We experiment with the following alternatives to define the replaced token distribution without an auxiliary model.

- *Uniform*: a uniform distribution defined over the entire vocabulary.

- *Term frequency*: a distribution where the probability mass of a token is equal to its frequency across the entire training corpus.
- *Smoothed one-hot*: a distribution where the probability mass of the correct token is $1 - \alpha$, while the probability mass of any other token is $\alpha/(|\mathcal{V}| - 1)$. Here $\mathcal{V}$ indicates the vocabulary and $\alpha = 0.35$ is the typical prediction error rate of an auxiliary model on the masked tokens.

Figure 3 visualizes the above distributions for an example sequence [3]. Note that the term frequency approximately follows Zipf's law (Newman, 2004). We also plot the replaced token distribution produced by an auxiliary model for reference.

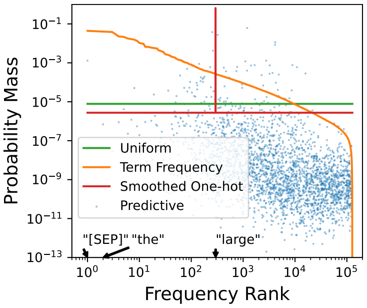

Figure 3: The probability distribution of the replaced token in an example sequence defined by various methods. The token classes are ranked by the term frequency in the x-axis.

We also include learning curriculums defined by these distributions by interpolation, as intuitively smoothed one-hot is more difficult than other distributions since it can be viewed as an auxiliary model that always predicts the original token correctly with high confidence (See more details in Appendix E.3).

Table 6 shows the downstream performance when pre-training with these replaced token distributions [4]. Pre-training with the uniform distribution diverges early at around 10K updates. We suspect this is because the replaced tokens sampled from a uniform distribution are too easy to detect since most of the tokens would be rare and unique considering a vast vocabulary. This is supported by our observation that the training loss of the discriminator plunges to 0 at the very beginning of the pre-training, after which the training diverges. In contrast, interpolating the uniform distribution with the more difficult smoothed one-hot distribution can converge smoothly. In fact, such an interpolation also yields better downstream performance than smoothed one-hot alone, which can be attributed to the benefit of the learning curriculum.

Interestingly, pre-training with term frequency alone can converge and yield reasonable downstream performance ($\sim 85.7\%$ MNLI Acc.), whereas interpolating term frequency and smoothed one-hot leads to consistently worse downstream performance than term frequency alone. We suspect that term frequency is superior because it better indicates the difficulty of possible replaced tokens in a context.

Finally, we note that pre-training with these auxiliary-model-free replaced token distributions are still inferior to auxiliary-model-based ones in terms of the downstream performance upon convergence.

Table 6: Downstream task performance (MNLI accuracy, average m/mm) when pre-training with different replaced tokens distributions.

Method	Fast-ELECTRA	Uniform	Term Frequency	Smoothed One-hot	UNF → SOH	TF → SOH
MNLI Acc.	89.1	Diverged	85.7	83.8	85.3	84.9

D.2 Does the Curriculum Matter for ELECTRA-style Pre-training?

In this section, we investigate whether a learning curriculum, namely a gradually more difficult RTD task, is necessary for ELECTRA-style pre-training. We focus on the case when an auxiliary model is available, given the inferior performance of model-free alternatives as mentioned above. The comparative experiment here is *fixed-auxiliary pre-training*, where the replaced token distribution is simply the output distribution of a pre-trained and fixed auxiliary model, without any modification.

Figure 4 (left) shows the downstream of pre-training with a fixed auxiliary model. In our experiments, for more than 10K training updates from the beginning, the main model's replaced token detection accuracy remains 0. Yet with more training updates it starts to converge and yield decent downstream performance ($\sim 88.3\%$ on MNLI).

[3] A quote from Clark et al. (2020), "*most current training methods require [large] amounts of compute to be effective*", where "*[*]*" indicates the token to be replaced.

[4] We use UNF → SOH and TF → SOH to denote uniform-to-smoothed one-hot interpolation and term frequency-to-smoothed one-hot interpolation respectively.

Nevertheless, an appropriate curriculum greatly improves training efficiency. Compared to fixed-auxiliary training, temperature-scaling this same auxiliary model improves the accuracy from 80% to 85% on MNLI when the number of training updates is limited (e.g., 2K). An appropriate curriculum can also advance the downstream performance upon convergence (88.4% vs. 89.1%).

Finally, we found that a learning curriculum is particularly important for auxiliary models with a large capacity. As shown in Figure 4 right, temperature-scaling the auxiliary model improves the MNLI accuracy for more than 3% compared to fixed-auxiliary training. This implies learning curriculum alleviates the sensitivity of the pre-training effectiveness to the auxiliary model's capacity.

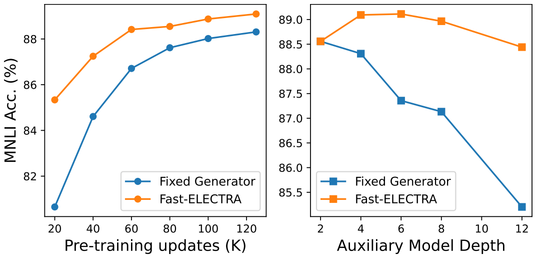

Figure 4: (*Left*): Downstream performance (MNLI accuracy, Avg m/mm) at multiple intermediate checkpoints, obtained by pre-training with a fixed auxiliary model and Fast-ELECTRA. (*Right*): Downstream performance at the last checkpoint versus the depth of the auxiliary model, obtained by pre-training with a fixed auxiliary model and Fast-ELECTRA.

E Additional Experiments

E.1 Training Efficiency

Table 7: Computation cost and memory consumption of ELECTRA measured on specific infrastructures

		4× RTX 3090		8× Tesla V100	
Model	Method	Computation (SPU)	Memory (GB)	Computation (SPU)	Memory (GB)
Base	Original	10.0	13.7	5.6	13.7
	Fast-ELECTRA	7.7	11.6	4.0	11.4
	Ratio	0.77	0.84	0.71	0.83
Large	Original	13.8	19.1	6.7	19.1
	Fast-ELECTRA	11.2	16.2	5.3	16.2
	Ratio	0.81	0.85	0.79	0.84

We test the overall computation cost and memory cost on specific computation infrastructures, as hardware-centered optimization can be critical to training efficiency (Rasley et al., 2020). We conduct experiments on two typical infrastructures, including a node with $4\times$ GeForce RTX 3090 GPUs (24GB memory each, w/o NVLink), and a node with $8\times$ Tesla V100 GPUs (32GB memory each, w/o NVLink). We measure the computation cost by the wall time (in seconds) per training update (SPU), and the memory cost by the peak memory occupied by all tensors on one GPU throughout training, averaged over all GPUs.

As shown in Table 7, Fast-ELECTRA can reduce the overall computation and memory cost of ELECTRA-style pre-training across different computation infrastructures and model settings consistently[5]. The reduction of computation cost matches our calculation in Section 4, while the reduction of memory cost is slightly less than that from our calculation, which is due to gradient accumulation that reduces the peak memory.

[5] Note that Fast-ELECTRA can in fact support a larger batch size per GPU and thus potentially speed up training further, due to reduced memory cost. Nevertheless, in our tests, we set the batch size per GPU to be the same for the original ELECTRA and Fast-ELECTRA, such that the computation reduction of Fast-ELECTRA involves no contribution of less memory cost.

E.2 Alternative Curriculum Designs for ELECTRA-style Pre-training

In this section, we explore alternative ways to design the learning curriculum when an auxiliary model is available. In general, a model-based curriculum can be determined by two functions, namely the schedule function and the augmentation function.

Augmentation function. An augmentation function is used to reduce the difficulty of the replaced token task generated by an existing auxiliary model. In Fast-ELECTRA, we utilized temperature scaling as an augmentation function to smooth the output distribution of the auxiliary model. Yet, essentially any method that can change the model's output distribution can be utilized as an augmentation function. Possible methods include changing the model's output distribution directly, changing the behaviors of the model weights or modules, or changing the model's input sequence. Here we consider the following alternative augmentation functions.

- *Logarithmic interpolation*: Interpolate the auxiliary model's output distribution with an easy distribution such as a uniform distribution or term frequency. Since interpolating with a uniform distribution is in fact equivalent to temperature scaling [6], we only consider the interpolation with term frequency, namely $p_{\text{TF}\to\theta}(\cdot|i, \boldsymbol{x}_{\text{masked}}) := p_{\text{TF}}^{\gamma} \odot p_{\theta}^{1-\gamma}(\cdot|i, \boldsymbol{x}_{\text{masked}})$, where $\odot$ denotes element-wise product [7].
- *Dropout*: Enable all activation dropout layers in the auxiliary model, and set all their drop rates as γ.
- *Drop attention*: Enable all attention dropout layers in the auxiliary model, and set all their drop rates as γ.
- *Drop token*: Randomly replace γ fraction of tokens (except special tokens such as "`[MASK]`") in the auxiliary model's input sequence with "`[UNK]`", namely the token representing unknown tokens.

Figure 5 left shows the downstream performance achieved by pre-training with these augmentation functions. Here for each augmentation, we use an exponentially decayed function to schedule γ similar to Equation 2, and search its best hyper-parameters (*i.e.*, $\gamma_{\max}$ and τ) based on the final performance. One may find that all these augmentation functions can achieve decent downstream performance. Nevertheless, in practice, we prefer temperature scaling as it is easy to implement and achieves slightly better performance.

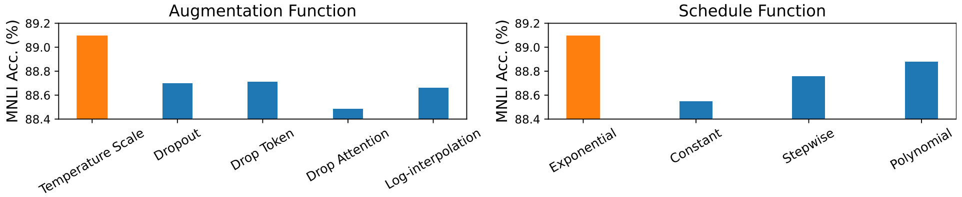

Figure 5: Downstream performance (MNLI accuracy, Avg m/mm) obtained by pre-training with different augmentation functions (*Left*) and different schedule functions (*Right*). The augmentation function and schedule function used in Fast-ELECTRA are highlighted.

Schedule function. A schedule function is used to determine the parameter of the augmentation function (*e.g.*, γ) at the u-th fraction of training updates, such that the difficulty of the replaced token detection task gradually increases through pre-training. In previous sections, we have experimented on an exponential decay function. Here we experiment on alternative schedule functions as follows.

- *Constant*: $\gamma(u) = \gamma_0$.
- *Polynomial decay*: $\gamma(u) = \gamma_{\max}(1-u)^{\tau}$, where τ here controls the decay rate and a larger τ results in a faster decay.

[6] Logarithmic interpolation with a uniform distribution is $\log p := \gamma \log p_{\text{UNF}} + (1-\gamma) \log p_\theta = -\gamma \log |\mathcal{V}| + (1-\gamma) \log p_\theta$, while temperature scaling can be formulated as $\log p := (1/T) \log p_\theta$. Therefore, these two will be the same up to a constant difference, which will be canceled after applying Softmax on the log probabilities.

[7] We choose to logarithmically interpolate the distributions because we empirically observed it is slightly better than linearly interpolating.

- *Stepwise decay*: $\gamma(u) = \gamma_{\max}(1 - \lfloor u\tau \rfloor/\tau)$, where $\lfloor \cdot \rfloor$ denotes the floor function and τ determines the number of decays.

Figure 5 right shows the downstream performance achieved by temperature-scaling with its parameter scheduled by these functions. For each schedule function, we search for its best hyper-parameter based on the final performance. One can find that the constant schedule leads to significantly lower performance than exponential decay, while other schedules can achieve comparable performance.

E.3 Learning Curriculum Free of Auxiliary Models

We experiment on curriculum RTD tasks defined by replaced token distributions free of auxiliary models. As mentioned in Section D.1, the smoothed one-hot distribution is intuitively more difficult than the uniform distribution and term frequency. Therefore, we can interpolate these distributions with a varied coefficient to gradually increase the difficulty of the RTD task during training. Concretely, we can define the replaced tokens distributions as

- Interpolation between uniform and smoothed one-hot (UNF→ SOH):

$$p_{\text{UNF}\to\text{SOH}} \coloneqq p_{\text{UNF}}^{\gamma} \odot p_{\text{SOH}}^{1-\gamma}.$$

- Interpolation between term frequency and smoothed one-hot (TF→ SOH):

$$p_{\text{TF}\to\text{SOH}} \coloneqq p_{\text{TF}}^{\gamma} \odot p_{\text{SOH}}^{1-\gamma}.$$

Here $\odot$ denotes the element-wise multiplication, and $0 \leq \gamma \leq 1$ and is scheduled by an exponentially decayed function, similar to Equation 2. We choose to logarithmically interpolate the distributions because we empirically observed it is slightly better than linearly interpolating.