Model-Generated Pretraining Signals Improves Zero-Shot Generalization of Text-to-Text Transformers

Linyuan Gong[1]*, **Chenyan Xiong**[2], **Xiaodong Liu**[2], **Payal Bajaj**[2], **Yiqing Xie**[3]*, **Alvin Cheung**[1], **Jianfeng Gao**[2] and **Xia Song**[2]
[1]UC Berkeley, [2]Microsoft, [3]Carnegie Mellon University
[1]{gly,akcheung}@berkeley.edu
[2]{chenyan.xiong,xiaodl,payal.bajaj,jfgao,xiaso}@microsoft.com
[3]yiqingxi@andrew.cmu.edu

Abstract

This paper explores the effectiveness of model-generated signals in improving zero-shot generalization of text-to-text Transformers such as T5. We study various designs to pretrain T5 using an auxiliary model to construct more challenging token replacements for the main model to denoise. Key aspects under study include the decoding target, the location of the RTD head, and the masking pattern. Based on these studies, we develop a new model, METRO-T0, which is pretrained using the redesigned ELECTRA-Style pretraining strategies and then prompt-finetuned on a mixture of NLP tasks. METRO-T0 outperforms all similar-sized baselines on prompted NLP benchmarks, such as *T0 Eval* and *MMLU*, and rivals the state-of-the-art $T0_{11B}$ model with only **8%** of its parameters. Our analysis on model's neural activation and parameter sensitivity reveals that the effectiveness of METRO-T0 stems from more balanced contribution of parameters and better utilization of their capacity. The code and model checkpoints are available at https://github.com/gonglinyuan/metro_t0.

1 Introduction

Recent work in NLP has shown that pretrained language models have made noteworthy progress toward generalization to unseen tasks. Despite being pretrained on only language modeling objectives, large language models can perform reasonable zero-shot generalization given natural language instructions, i.e. prompts (Radford et al., 2019; Brown et al., 2020). Further research shows that finetuning language models on a mixture of tasks with prompt templates enhances their performance on held-out new tasks (Sanh et al., 2022; Wei et al., 2021).

In recent years, two significant research paths have emerged in the field of pretrained language models: one seeks to improve generalization either by scaling up the model, increasing parameters, data, and compute, or by refining prompts. Another divergent yet complementary approach focuses on augmenting the efficiency of pretraining, particularly in the context of BERT-style models. This approach has been proven to significantly improve pretraining efficiency through the use of model-generated pretraining signals, as evidenced by ELECTRA (Clark et al., 2020), COCO-LM (Meng et al., 2021), and METRO-LM (Bajaj et al., 2022). However, this improvement has primarily been witnessed in single-task supervised finetuning settings. Our work seeks to bridge these two areas of research. We present a novel method that enhances the pretraining efficiency of T5, a widely used encoder-decoder Transformer in prompt-based learning, by utilizing ELECTRA-Style model-generated signals.

Our preliminary studies, however, encountered many challenges in pretraining T5 with model-generated signals, particularly in designing an effective objective to train the decoder and ensuring training stability. To address these challenges, we study the impact of key components in this pretraining scheme, such as the decoding target, the location of the Replace Token Detection (RTD) task, and the masking pattern. Then we redesign the pretraining algorithm to solve training stability issues, thus bringing in the benefits of ELECTRA-style pretraining to T5-style Transformer encoder-decoder models. The pretrained model is then finetuned on a family of multi-task training mixtures of NL-prompted dataset, which has previously been used to train the T0 models (Sanh et al., 2022). Our model, METRO-T0, is a *T0* model pretrained with Model generated dEnoising TRaining Objective.

Experimental results show that METRO-T0 is highly *parameter efficient*. It consistently outperforms similar-sized baselines on all NL-prompted benchmark we evaluated upon. As shown in Fig-

*Part of this work is done during Linyuan and Yiqing's internship at Microsoft.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 12933–12950

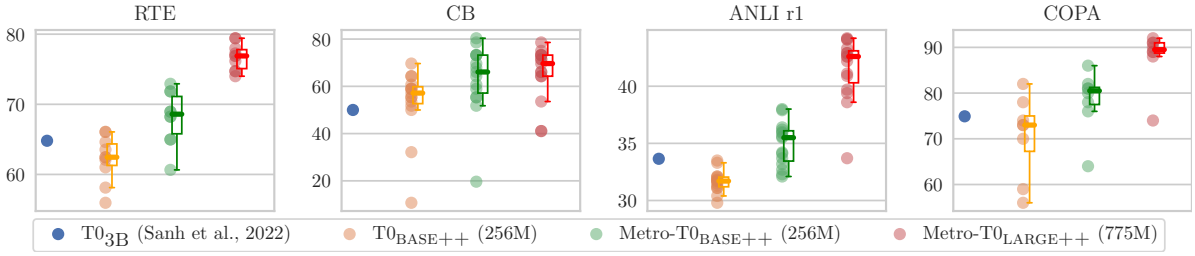


Figure 1: Prompt learning results of METRO-T0 versus our T0 baseline and T0$_{3B}$ by Sanh et al. (2022) on 4 tasks in the *T0 Eval* benchmark. Each point denotes the accuracy using one prompt template, except that the median accuracy over all templates of T0$_{3B}$ is indicated by the **blue** point. The plots of other tasks are in Appendix A.7.

ure 1, METRO-T0$_{BASE++}$ outperforms T0$_{3B}$ (Sanh et al., 2022) with only 7% of its parameters on the *T0 Eval* benchmark. Moreover, METRO-T0++$_{LARGE++}$ rivals 14x larger T0++$_{11B}$, the state-of-the-art in prompt-based learning. Our method is also *compute efficient*: METRO-T0 pretrained for 500k steps has similar performance as its T0 counterpart pretrained for 2M steps.

To further understand the benefit of METRO pretraining, we conduct two studies on the pretrained METRO-T0 model, analyzing its neural activation and parameter sensitivity. The studies show that model-generated signals balance the contribution of each NN parameter and reduce the number of under-activated neurons by 55%, indicating that a key source of the improved pretraining efficiency is better utilization of network parameters.

2 Related Work

Prompt-based learning with language models. *Prompt-based learning* allow language models to handle a wide range of tasks with no training data (zero-shot) or a few training data (few-shot), by leveraging natural language instructions and task demonstrations as context (Radford et al., 2019; Brown et al., 2020). Raffel et al. (2019) proves the effectiveness of prompt-based learning as a framework of multi-task learning for text-to-text Transformers such as T5. LMs are usually finetuned with NL instructions to improve their performance and usability. Such a procedure is called prompt-finetuning. The finetuning data comes from aggregated mixtures of NLP tasks (Sanh et al., 2022; Wei et al., 2021), dialogs (Chung et al., 2022), or even chain-of-thoughts (Wei et al., 2022). Our work aims to improve the zero-shot generalization of T5-like text-to-text LMs in prompt-based learning by efficient and effective pretraining strategies.

Efficient pretraining using model-generated signals. Training big language models require substantial computational resources. This paper is part of a line of research that improves the pretraining efficiency of LMs using model-generated signals, i.e., METRO (Bajaj et al., 2022), pioneered by ELECTRA (Clark et al., 2020), a Transformer encoder pretrained using signals generated by an auxiliary BERT. Various studies (Meng et al., 2021, 2022; Chi et al., 2021; Fang et al., 2022) show that an auxiliary model can generate informative training signals that greatly improve the efficiency and effectiveness of BERT-like Transformer *encoder* models, as evaluated on supervised single-task benchmarks like GLUE (Wang et al., 2018). Compared with these works, we use model-generated signals to pretrain T5-like Transformer *encoder-decoder* models and evaluate this model on large-scale NL-prompted benchmarks.

3 Preliminaries

This section provides an overview of T5 and METRO-style pretraining.

3.1 Text-to-Text Transformers

Our models are based on the T5 framework (Raffel et al., 2019). T5 is a text-to-text Transformer pretrained on natural language corpus.

T5 Pretraining. T5 is a Transformer encoder-decoder language model pretrained by modeling corrupted spans of subword tokens. The noisy input is constructed by replacing consecutive spans of tokens in the input by distinct "sentinel" tokens, e.g., $X^{\text{noise}} = [x_1^{\text{orig}}, ..., \texttt{[M]}^{i:j}, ...x_n^{\text{orig}}]$, where the sentinel token is denoted by $\texttt{[M]}^{i:j}$. Then the pretraining task is to generate the deleted tokens using the Transformer decoder, conditioned on X^{noise} as input to the Transformer encoder:

$$[x_1^{\text{orig}}, \ldots \texttt{[M]}^{i:j}, \ldots x_n^{\text{orig}}] \xrightarrow{\text{Encoder}} \boldsymbol{H}^{\text{enc}}$$
$$\boldsymbol{H}^{\text{enc}} \xrightarrow{\text{Decoder}} [\texttt{[M]}^{i:j}, x_i^{\text{orig}}, ..., x_j^{\text{orig}}]. \quad (1)$$

Text-to-Text Formulation of Downstream Tasks. T5 supports multitask learning on a diverse set of downstream tasks—including classification, question answering, and summarization—by casting all these tasks into a *text-to-text* format, where the encoder is fed with the text input and the decoder is then asked to generate the target prediction.

Text-to-Text Prompt-Finetuning. A pretrained text-to-text Transformer can then be finetuned to enhances its performance on held-out new tasks. The finetuning corpus is usually a multi-task mixture of NLP datasets, where each input-output pair is an example formatted with an NL prompt template. The finetuning procedure is standard seq2seq learning: the input sequence is fed to the encoder, and the target sequence serves as the ground truth to compute the cross-entropy loss of the decoder output.

3.2 Model-Generated Pretraining Signals

In this subsection, we discuss techniques involving model-generated pretraining signals in prior work.

Replace token detection (RTD) is the training objective used to train ELECTRA (Clark et al., 2020). The RTD input is a noisy text sequence X^{noise}, generated by an auxiliary masked language model (MLM) like BERT. The token x_i^{noise} in each masked position of the text sequence is sampled from the predicted probability of the auxiliary model $p_{\text{MLM}}(x_i^{\text{noise}}|\mathbf{h}_i^{\text{aux}})$, while the token in each unmasked position x_j^{noise} is copied from the original text x_j^{orig}. The main model, a Transformer encoder, is pretrained to denoise the noisy input by classifying whether each token is replaced by the auxiliary model or from the original text.

$$X^{\text{orig}} \xrightarrow{\text{Random Mask}} [x_1^{\text{orig}}, \dots \texttt{[M]}, \dots x_n^{\text{orig}}]; \quad (2)$$

$$[x_1^{\text{orig}}, \dots \texttt{[M]}, \dots x_n^{\text{orig}}] \xrightarrow{\text{Auxiliary}} X^{\text{noise}}; \quad (3)$$

$$X^{\text{noise}} \xrightarrow{\text{Model}} \boldsymbol{H} \xrightarrow{\text{RTD Head}} \mathbb{1}(x_i^{\text{orig}} = x_i^{\text{noise}}). \quad (4)$$

Prior work show that the RTD objective is more efficient than the MLM objective, resulting in significant performance improvement for pretrained Transformer encoders (Clark et al., 2020). However, replacing MLM with RTD turns the generative model into a discriminative model, hindering the model's ability to perform generation.

Corrective language modeling (CLM) restores the generation capability of a Transformer encoder model pretrained with RTD (Meng et al., 2021). The CLM objective is trained alongside the RTD objective in a multi-task manner, so the CLM input is the same as the RTD input X^{noise}. The model is pretrained to recover the original text X^{orig}.

$$X^{\text{noise}} \xrightarrow{\text{Model}} \boldsymbol{H} \xrightarrow{\text{CLM Head}} X^{\text{orig}}. \quad (5)$$

4 Method

In this section, we present the algorithm to train our model, METRO-T0.

4.1 Pretraining Objective Design

METRO-T0 is jointly pretrained with two objectives: the RTD objective, enhancing performance through model-generated signals, and the CLM objective, enabling text-to-text generation akin to T5. The pretraining algorithm is illustrated in Figure 2. METRO-T0 uses a BERT-style MLM encoder as the auxiliary model and a T5-style encoder-decoder as the main model. The overall pretraining procedure is:

$$X^{\text{orig}} \xrightarrow{\text{i.i.d. Random Mask}} [x_1^{\text{orig}}, \dots \texttt{[M]}, \dots x_n^{\text{orig}}]; \quad (6)$$

$$[x_1^{\text{orig}}, \dots \texttt{[M]}, \dots x_n^{\text{orig}}] \xrightarrow{\text{Auxiliary}} X^{\text{noise}}; \quad (7)$$

$$X^{\text{noise}} \xrightarrow{\text{Encoder}} \boldsymbol{H}^{\text{enc}} \xrightarrow{\text{RTD Head}} \mathbb{1}(x_i^{\text{orig}} = x_i^{\text{noise}}); \quad (8)$$

$$\boldsymbol{H}^{\text{enc}} \xrightarrow{\text{Decoder}} \boldsymbol{H}^{\text{dec}} \xrightarrow{\text{CLM Head}} X^{\text{orig}}. \quad (9)$$

The auxiliary model receives inputs constructed by randomly masking tokens in the original text X^{orig}, and makes MLM predictions, which are used to create noisy inputs X^{noise} for the main model. The main model is pretrained using two objectives: **(a)** the RTD objective on the encoder outputs $\boldsymbol{H}^{\text{enc}}$, which aims to identify whether each token was replaced by the auxiliary model or not, and **(b)** the CLM objective, which aims to recover the original text X^{orig} through the decoder. During pretraining, the weighted average of three losses is optimized:

$$\mathcal{L}_{\text{MLM}} = -\mathbb{E}_{i\in\mathcal{M}} \log p_{\text{MLM}}(x_i^{\text{orig}}|\mathbf{h}_i^{\text{aux}}), \quad (10)$$

$$\mathcal{L}_{\text{RTD}} = -\mathbb{E} \log p_{\text{RTD}}(\mathbb{1}(x_i^{\text{orig}} = x_i^{\text{noise}})|\mathbf{h}_i^{\text{enc}}), \quad (11)$$

$$\mathcal{L}_{\text{CLM}} = -\mathbb{E}_{i\in\mathcal{M}} \log p_{\text{LM}}(x_i^{\text{orig}}|\mathbf{h}_i^{\text{dec}}), \quad (12)$$

$$\mathcal{L} = \mathcal{L}_{\text{MLM}} + \lambda_{\text{RTD}}\mathcal{L}_{\text{RTD}} + \lambda_{\text{CLM}}\mathcal{L}_{\text{CLM}}. \quad (13)$$

In crafting METRO-T0's pretraining algorithm, we explored various alternatives before finalizing our design. For example, an alternative method

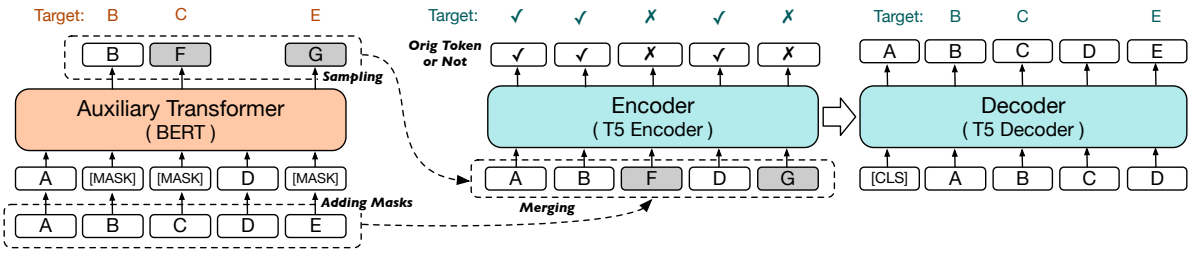


Figure 2: The architecture of METRO-T0 during pretraining using BERT as the auxiliary model to generate signals.

Original Sentence		Thank you for inviting me to your party last week
Auxiliary Model	*Input*	Thank you `[M]` `[M]` me to your party `[M]` week
	Output	for giving apple
Main Model	*Input*	Thank you for giving me to your party apple week
Decoding Target	*Masked Tokens Only*	for inviting last
	All Tokens	Thank you for inviting me to your party last week
	All Tokens, Masked Loss★	Thank you for inviting me to your party last week

Table 1: Examples of encoder inputs and decoder targets of different ways to configure the denoising task. `[M]` denotes a shared mask token. The auxiliary MLM model predicts one token for each `[M]`. Grayed-out tokens are part of the target fed into the decoder but not included in pretraining loss.

could train RTD objectives on *decoder* outputs or use a masking pattern other than i.i.d. random sampling. In the rest of this section, we will explain our design choices and the reasons behind them.

Decoding Target. Table 1 shows three variants of decoding targets: *"masked tokens only"*, *"all tokens"*, and *"all tokens masked loss"*.

Pretraining with the T5-style *"masked tokens only"* target proves unfeasible due to its ill-formed nature. The decoder cannot distinguish between unmasked tokens (e.g., *"you"*) and those correctly predicted by the auxiliary model in masked positions (e.g., *"for"*). Consequently, a single source sequence may correspond to multiple correct target sequences, introducing ambiguity and impeding effective pretraining. A detailed example is provided in Appendix A.9.

The *"all tokens"* target is inefficient, as the cross entropy loss is averaged on all tokens, including unmasked tokens where the tasks are trivial copy-and-pastes. Therefore, METRO-T0 uses *"all tokens masked loss"*, where the loss is averaged on masked tokens only.

Location of the RTD Head. We consider two choices to place the RTD head: on the outputs of the Transformer *encoder* or *decoder*. *Decoder RTD* at position i requires the information of the i-th token of the encoder input, but this information is absent from the input of the decoder. Consequently, the decoder needs a long attention path to connect position i of the encoder. This complexity defeats the purpose of RTD in providing a simpler task to stabilize optimization, making pretraining unstable in practice (Xu et al., 2020). Therefore, METRO-T0 uses *encoder RTD*.

Masking Pattern on Auxiliary. When can use either T5-style *contiguous span masking* or BERT-style *i.i.d. random masking* to generate the MLM input for the auxiliary model. However, using *contiguous span masking* in METRO-T0 pretraining leads to label leakage. At position i during teacher-forced training, the decoder has access to the ground truth X^{orig} before position i. It can compare x_{i-1}^{orig} with x_{i-1}^{noise}. If the two disagree, it is likely the following position i is also masked out. As a result, the model can exploit this shortcut to achieve high RTD accuracy without learning meaningful representations of natural languages. Therefore, METRO-T0 uses *i.i.d. random masking*.

4.2 Architectural Upgrades over T5

We incorporate model architecture changes that have been proved to be beneficial in earlier works.

The vanilla T5 exclusively uses relative positional embeddings, while the vanilla BERT (Devlin et al., 2019) model relies solely on absolute positional embeddings. However, recent research by Luo et al. (2022) suggests that using only relative positional embeddings may not yield optimal results. Consequently, in line with the practices in COCO-LM (Meng et al., 2021) and METRO-

LM (Bajaj et al., 2022), we use absolute positional embeddings in addition to relative position embeddings in our model.

We also introduce a change in how layer normalization is combined with residual connections. Rather than using T5's Pre-LayerNorm approach (defined as $x \mapsto x + f(\mathrm{LN}(x))$ where f is either multi-head attention or MLP), our model adopts a Post-LayerNorm design ($x \mapsto \mathrm{LN}(x + f(x))$). The Post-LayerNorm vs. Pre-LayerNorm debate is ongoing in the field, but we use Post-LayerNorm, which typically resulted in better performance on downstream tasks in our studies.

4.3 Prompt-Finetuning

The model pretrained using the method described above is called METRO-T5. After pretraining METRO-T5 on an NL corpus, we discard the auxiliary model and retain the main model, which is a standard text-to-text Transformer. We finetune this model on multi-task training mixtures of NL-prompted datasets, *T0/T0+/T0++ Train* (Sanh et al., 2022), to obtain METRO-T0/T0+/T0++, a text-to-text Transformer that supports zero-shot generalization to held-out tasks.

5 Experimental Setup

Model Architecture. Each of our models has an architecture similar to T5 (Raffel et al., 2019). We train models in three standard setups: *base*, *base++*, and *large++*. Our *base*/*base++* model has an architecture similar to $\mathrm{T5}_{\mathrm{BASE}}$. Our *large++* model has an architecture similar to $\mathrm{T5}_{\mathrm{LARGE}}$ except for some differences mentioned in Section 4. The auxiliary model for generating training signals is a Transformer encoder of the same hidden size as the main model but is shallower: it consists of 4 layers in *base*/*base++* and 6 layers in *large++*. We follow Clark et al. (2020) and share token embeddings between the main and the auxiliary model.

Pretraining. Our *base* model is pretrained on English Wikipedia and BookCorpus (16GB of texts) for 131 billion tokens (512 tokens per sequence, 2,048 sequences per batch, and 125k steps). *Base++*/*Large++* is the training configuration first used in RoBERTa (Liu et al., 2019): pretraining on a mixed corpus of 160GB texts for a maximum 2.1 trillion tokens (512 tokens per sequence, 2,048 sequences per batch, and at most 2M steps).

Prompt-Finetuning. We finetune each of our pretrained METRO-T5 models on three multi-task mixtures: *T0/T0+/T0++ Train*, using the same prompt templates and shuffling strategy as Sanh et al. (2022) does. Each model is finetuned for 125k steps, using the same hyperparameters as pretraining, except the peak learning rate is reduced to 0.1x. We do not perform any checkpoint selection and simply use the last checkpoint at 125k steps for evaluation.

Evaluation. We evaluate zero-shot generalization on the *T0 Eval* benchmark (Sanh et al., 2022) and the *Massive Multi-task Language Understanding* (*MMLU*) benchmark (Hendrycks et al., 2020). *T0 Eval* consists of 11 datasets in natural language inference, coreference, word sense disambiguation, and sentence completion. *MMLU* includes exam questions from 57 tasks such as maths, history, law, and medicine. For each dataset, we report accuracy on the validation split. Following GPT-3 (Brown et al., 2020) and T0 (Sanh et al., 2022), we use rank classification for inference.

For *T0 Eval*, we use the same prompt templates as T0. For MMLU, we use prompt templates from the *AI2 Reasoning Challenge* (*AI2-ARC*) (Clark et al., 2018), concatenated with 5 passages retrieved using T5-ANCE (Ge et al., 2023; Ni et al., 2021) (See Appendix A.8 for details). When there are multiple prompts for a dataset, we do not perform prompt selection based on the validation split, because such prompt selection will break the "zero-shot" evaluation. Instead, we report the average accuracy across all prompts for this dataset, following the standard practices of Sanh et al. (2022).

Baselines. For a fair comparison, the main baseline is our own T0 runs. Except for METRO-style pretraining, our T0 baselines use the same Transformer architecture, pretraining data, and prompt-finetuning data, pretrained in the same computational environment.

We also compare with the *reported* numbers of other language models that supports zero-shot prompting, including pretraining-only models such as GPT-3 (Brown et al., 2020) and T5 (Raffel et al., 2019), as well as prompt-finetuned models such as T0 (Sanh et al., 2022) and Flan-T5 (Wei et al., 2021; Chung et al., 2022). T0/T0+/T0++ is pretrained on the the C4 (Raffel et al., 2019) corpus of 800GB of texts for 1 trillion tokens and then prompt-finetuned on the *T0/T0+/T0++ Train* multitask mixture af-

Model	Params	NLI			Coref.		Compl.			WSD	
		RTE	CB	ANLI r1/r2/r3	WSC	Wino.	COPA	SC.	HS.	WiC	AVG
Pretraining only											
GPT-3_{13B} (Brown et al., 2020)	13B	62.80	19.60	33.20/33.50/34.40	64.40	67.90	84.00	79.50	70.90	0.00	50.02
GPT-3_{175B} (Brown et al., 2020)	175B	63.50	46.40	34.60/35.40/34.50	65.40	70.20	91.00	83.20	78.90	0.00	54.83
T5+LM (Lester et al., 2021)	11B	53.03	34.34	32.89/33.76/33.82	54.09	50.65	54.88	27.00	48.16	50.30	42.99
Prompt Finetune on *T0 Train*											
$T0_{BASE}$	226M	62.85	45.30	30.82/32.37/32.14	**62.16**	50.77	70.63	**81.03**	24.86	**50.78**	49.43
METRO-$T0_{BASE}$	226M	**65.18**	**45.60**	**31.64/32.98/33.81**	55.77	**51.07**	**70.81**	80.97	**25.28**	50.69	**49.44**
$T0_{BASE++}$	256M	62.24	53.45	31.68/32.94/34.88	**61.73**	51.65	70.63	87.62	25.88	**51.21**	51.26
METRO-$T0_{BASE++}$	256M	**68.16**	**63.21**	**34.92/33.81/36.82**	60.48	**52.03**	**78.50**	**89.23**	**27.68**	50.88	**54.15**
$T0_{3B}$ (Sanh et al., 2022)	3B	64.55	45.36	33.84/33.11/33.33	**65.10**	50.97	72.40	84.03	27.29	50.69	50.97
METRO-$T0_{LARGE++}$	775M	**76.75**	**65.48**	**41.49/36.29/40.18**	60.58	**54.51**	**88.00**	**94.07**	**29.31**	**50.97**	**57.97**
$T0_{11B}$ (Sanh et al., 2022)	11B	80.83	70.12	43.56/38.68/41.26	61.45	59.94	90.02	92.40	33.58	56.58	60.77
Prompt Finetune on *T0+ Train*											
$T0+_{BASE}$	226M	63.57	**48.93**	31.76/32.92/33.02	**60.96**	**51.93**	**72.38**	*81.71*	*40.11*	**51.32**	51.69
METRO-$T0+_{BASE}$	226M	**70.56**	47.08	**33.05/34.53/34.37**	57.98	51.75	69.13	***83.08***	***49.00***	50.78	**52.85**
$T0+_{BASE++}$	256M	68.30	60.24	33.77/34.31/35.00	60.96	51.59	70.00	*89.29*	*56.10*	51.39	55.54
METRO-$T0+_{BASE++}$	256M	**71.44**	**60.71**	**36.91/35.24/36.46**	**62.21**	**54.08**	**78.88**	***90.29***	***67.57***	**51.60**	**58.67**
METRO-$T0+_{LARGE++}$	775M	81.26	70.00	45.06/38.59/42.35	60.67	57.52	90.50	*95.41*	*83.82*	52.32	65.23
$T0+_{11B}$ (Sanh et al., 2022)	11B	67.47	59.20	43.45/39.77/40.76	62.24	59.94	92.24	*96.43*	*86.13*	55.02	63.88
Prompt Finetune on *T0++ Train*											
$T0++_{BASE}$	226M	69.06	48.39	31.90/33.61/33.94	*55.72*	*51.15*	***76.06***	*82.55*	*39.62*	*63.18*	53.20
METRO-$T0++_{BASE}$	226M	**72.04**	**58.63**	**33.85/35.29/36.57**	***56.11***	***52.15***	*74.06*	***83.65***	***48.66***	***64.29***	**55.94**
$T0++_{BASE++}$	256M	**77.87**	63.10	36.15/34.61/38.18	*56.44*	*51.78*	*75.38*	*89.33*	*55.95*	*65.53*	58.57
METRO-$T0++_{BASE++}$	256M	77.80	**69.52**	**39.69/36.61/40.08**	***61.44***	***54.55***	***83.88***	***90.88***	***68.54***	***67.59***	**62.78**
METRO-$T0++_{LARGE++}$	775M	83.68	74.88	46.84/40.37/44.95	*71.83*	*62.75*	*92.63*	*95.65*	*83.74*	*70.49*	69.80
$T0++_{11B}$ (Sanh et al., 2022)	11B	85.31	75.69	47.07/42.18/44.09	*70.29*	*66.42*	*93.71*	*96.49*	*86.11*	*70.02*	70.67

Table 2: Prompt learning results on the *T0 Eval* dataset. "Wino.", "SC.", and "HS" refer to Winogrande, StoryCloze, and HellaSwag, respectively. All reported datasets use accuracy as their metric. *Italic* results are produced under the supervised setting. Others are under the zero-shot setting. Each row without a citation contains experimental results from models trained by us (our T0 baseline and METRO-T0), while each row with a citation contains experimental results from the cited paper (GPT-3, Google T5, and the original T0).

Model	Params	MMLU
$T0++_{BASE}$	226M	37.5
METRO-$T0++_{BASE}$	226M	**38.3**
Flan-$T5_{BASE}$ (Wei et al., 2022)	223M	35.9
$T0++_{BASE++}$	256M	41.7
METRO-$T0++_{BASE++}$	256M	**42.7**
GPT-3_{175B} (Brown et al., 2020)	175B	43.9
Flan-$T5_{LARGE}$ (Wei et al., 2022)	750M	45.1
$T0++_{11B}$ (Sanh et al., 2022)	11B	35.6
METRO-$T0++_{LARGE++}$	775M	**48.0**

Table 3: Prompt learning results on the *MMLU* dataset. All reported results use accuracy averaged over 57 subtasks as their metric.

ter LM adaptation for 100 billion tokens. Flan-T5 is also pretrained on the C4 corpus, but finetuned on a much larger dataset of prompted multi-task mixtures, dialog, and chain-of-thoughts.

6 Evaluation Results

This section compares the performance of METRO-T0 and baseline models on *T0 Eval* and *MMLU* to demonstrate the effectiveness and efficiency of our method. We also explore the reason behind METRO-T0's effectiveness through detailed model analysis.

6.1 Main Results

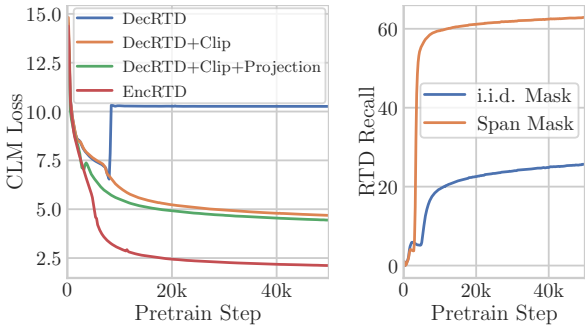
Table 2 presents the experimental results on *T0 Eval*, and Table 3 presents the experimental results on *MMLU*. These results show that:

METRO-T0 is highly parameter efficient, as it rivals or even outperforms much larger models in zero-shot generalization. METRO-$T0_{BASE++}$, having only 256M parameters, outperforms $T0_{3B}$ (Sanh et al., 2022) with only **7%** of its parameters. Also, METRO-$T0_{LARGE++}$, having only 775M parameters, outperforms $T0_{3B}$ by 7pts and is only 2.8pts behind $T0_{11B}$, a 14x larger model.

METRO-T0 often outperforms GPT-3 (175B), a state-of-the-art Transformer decoder LM, on both *T0 Eval* and *MMLU*. Compared to the 11B-parameter T0/T0+/T0++ model, a family of

Model/Finetuning Data	T0	T0+	T0++
METRO-T0/T0+/T0++	49.44	**54.15**	**57.97**
+ CLM Loss on All Position	49.24	51.05	53.97
+ CLM with Copy Mechanism	**49.46**	50.70	54.06
+ RTD on Decoder	46.75	48.47	49.20
+ Projection Layer on CLM	48.85	50.10	52.82
+ Continuous Span Mask	49.04	50.37	53.42
T0/T0+/T0++	**49.43**	**51.69**	**53.20**
+ All-token LM loss	48.13	49.43	50.76

Table 4: Performance of METRO-T0 variations on *T0 Eval*. All ablations are done in the *base* pretraining setting using exactly the same prompt-finetuning pipeline.



(a) RTD Position (b) Masking Strategy

Figure 3: Pretraining behaviors of different designs.

state-of-the-art prompt-finetuned text-to-text LM, METRO-T0/T0+/T0++ in the *large*++ setup has competitive or sometimes superior performance.

The gain stems from METRO-style pretraining. On both benchmarks, METRO-T0 models in all setups consistently outperform our fair-comparison T0 baselines of the same model size, which were pretrained using the same corpus and configurations. This fact demonstrates that the performance improvement is not due to better hyperparameters or data engineering, but a result of using METRO-style pretraining. Further confirmation of this argument will be provided through model analysis in Section 6.4 and Section 6.5.

6.2 Ablation Studies

In Section 4, we discuss the choices we made to redesign the pretraining method for METRO-T0. In this subsection, we compare the empirical results of different variants of METRO-T0. Table 4 shows the performance of each variant prompt-finetuned on *T0/T0+/T0++ Train* and evaluated on *T0 Eval*.

"All tokens, masked loss" is the best decoding target. Table 1 presents three possible choices for the decoding target, in which *"masked tokens only"* is ill-formed and thus not suitable, as discussed in Section 4. Table 4 compares the remaining two options and shows that computing CLM/LM loss on all positions negatively affects the downstream performance of METRO-T5/T5 by overwhelming the model with too many trivial copy-and-paste tasks. The same reasoning also applies to our decision not to use the copy mechanism (Meng et al., 2021) in CLM heads.

Encoder RTD makes pretraining more stable. Figure 3a demonstrates this by comparing the loss on the CLM task during pretraining with RTD applied to the encoder (**red** line) versus the decoder (**blue** line). Decoder RTD caused pretraining to diverge. While techniques such as strong gradient clipping and an additional projection layer can mitigate this issue (**orange** and **green** lines), the model still has higher training loss and poorer generalization on downstream tasks as shown in Table 4.

Label leakage is prevented by i.i.d. masking. Figure 3b illustrates the RTD *recall* (true positive rate) of METRO-T5 when using i.i.d. random masking on the auxiliary model compared to T5's continuous span masking. As discussed in Section 4, continuous span masking leads to label leakage, resulting in easy solutions for many masked positions, as demonstrated by the more than 2x pretraining RTD recall on masked positions with **Span Mask**. As expected, this label leakage hurts the model's generalization ability as shown in Table 4.

6.3 Pretraining Efficiency

In this experiment, we study the pretraining efficiency of METRO-T5 by comparing the intermediate checkpoints pretrained for 500k/1M/2M steps of $\text{T5}_{\text{BASE++}}$ and $\text{METRO-T5}_{\text{BASE++}}$. We assess each checkpoint's prompt-based learning performance by finetuning on the *T0++ Train* dataset and recording the average performance on *T0 Eval*.

Figure 4 shows that **METRO-T5 is more compute efficient than vanilla T5**. METRO-T0++ achieves better downstream performance at *every point*. In particular, METRO-T0++ pretrained for 500k steps has a similar performance to T0++ pretrained for 2M steps, showing a **165%** efficiency increase.

An interesting research question is: does model-generated signals simply make pretraining faster or do METRO-T5 and T5 learn different representations?

To answer this question, we compare the following two models by showing their performance on

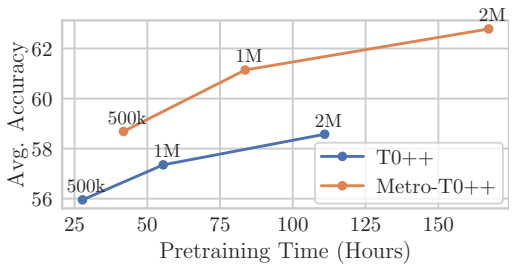


Figure 4: Comparison of the pretraining efficiency of T5 and METRO-T5. Each point shows the performance of a T0++/METRO-T0++ model finetuned from a checkpoint at 500k/1M/2M pretraining steps. The x-axis displays the pretraining wall time, reflecting computational cost, as all models were pretrained in the identical environment.

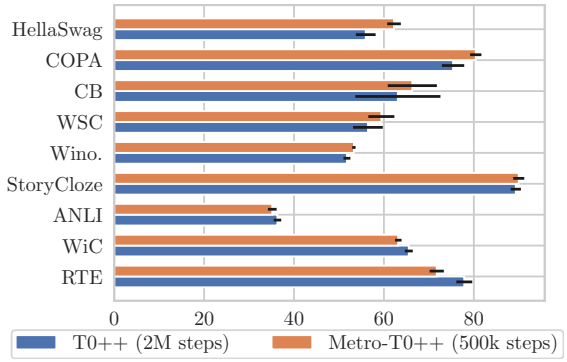


Figure 5: Per-task performance of T0++ (pretrained for 2M steps) and METRO-T0++ (pretrained for only 500k steps) on *T0 Eval*. The error bars are calculated using the model's performance across prompt templates.

each task in the *T0 Eval* benchmark in Figure 5: **(a)** T0++ finetuned from the T5 checkpoint pretrained for 2M steps, indicated by the last **blue** datapoint in Figure 4; **(b)** METRO-T0++ finetuned from the METRO-T5 checkpoint pretrained for 500k steps, indicated by the first **orange** datapoint. Although these two models have similar average accuracies (58.57 vs. 58.68), they have different strengths, as shown in Figure 5. T0++ (2M steps) outperforms METRO-T0++ (500k steps) on word-level tasks (WiC) and conventional natural language inference (ANLI and RTE), while METRO-T0++ (500k steps) has much better performance on commonsense reasoning (HellaSwag and COPA). This phenomenon implies that model-generated signals let the model learn different representations of texts, which finally result in a significant performance gap between the fully pretrained T0++ and METRO-T0++, as shown in Table 2.

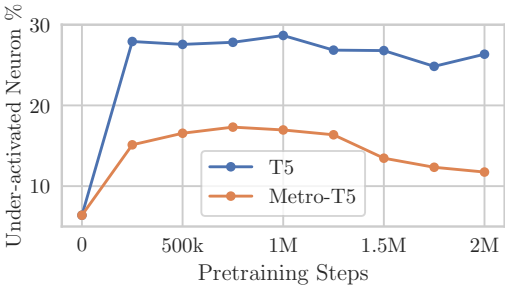


Figure 6: Comparison of the percentage of under-activated neurons in T5 and METRO-T5 on *T0++ train* dataset. The first point of both models (0 steps) overlap because they are the same initial model.

6.4 Neural Activation

In this subsection, and the following one, explore the extent to which the internal statistics of the neural networks quantify the differences between METRO-T5 and T5.

The first aspect we explore is neural activation. Specifically, we examine the feedforward module in each Transformer layer of METRO-T5$_{\text{BASE++}}$ and T5$_{\text{BASE++}}$, counting neurons that are *under-activated*. A neuron is considered *under-activated* if it is *inactive* (exhibits zero ReLU activations) for 99.5% of tokens within the *T0++ Train* dataset.

Figure 6 shows that T5 has ∼**2x** as many under-activated neurons as METRO-T5 at *every* checkpoint. Studies suggest that such neurons can typically be pruned without substantially affecting neural network performance (Polyak and Wolf, 2015; Li et al., 2016). So the presense of many under-activated neurons is a sign of underutilization of model capacity and computing cost. Therefore, our findings suggest that METRO-style model-generated training signals enhance neuron utilization in METRO-T5.

6.5 Parameter Sensitivity

In addition to analyzing the neural activation of T5 and METRO-T5, we also examine their parameter sensitivity, which serves as another means to quantify the underlying differences between T5 and METRO-T5.

The *sensitivity* of a parameter, defined in Equation (14), approximates the change in the loss magnitude when this parameter is completely zeroed-out. θ denotes the parameter vector and $\mathcal{L}$ denotes the loss function. θ_{-j} denotes the parameter vector θ with its j-th entry set to zero. The approximation is derived from the first-order Taylor expansion of $\mathcal{L}$ at θ. Therefore, the sensitivity of the j-th param-

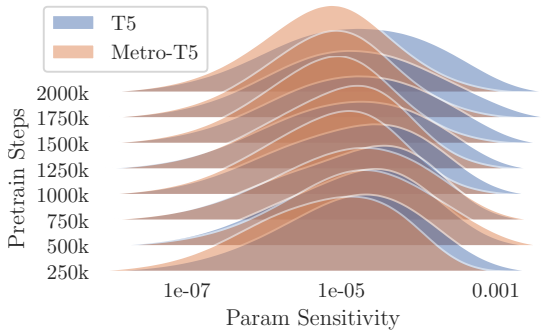


Figure 7: Comparison of the parameter sensitivity distributions of T5 and METRO-T5. Each row shows the parameter sensitivity distributions of T5 and METRO-T5 at the same pretraining step, indicated by the corresponding label on the y-axis.

eter, denoted by I_j, approximates the change in the loss magnitude when this parameter is completely zeroed-out (LeCun et al., 1989).

$$I_j = |\theta_{-j}^T \nabla_\theta \mathcal{L}(\theta)| \approx |\mathcal{L}(\theta) - \mathcal{L}(\theta - \theta_{-j})| \quad (14)$$

Liang et al. (2022) shows that parameter sensitivity is a reliable indicator of redundancy in pretrained language models. Specifically, parameters with low sensitivity can be safely pruned with only marginal impact on the LM's downstream performance, and an LM with lower, more concentrated sensitivity is more sufficiently trained and generalizes better.

We compare parameter sensitivity distributions of each checkpoint of METRO-T5 and T5, using gradients calculated on the *T0++ Train* dataset. The result is shown in Figure 7, from which we observe that the sensitivity distribution exhibits a lower variance in METRO-T5 (the **orange** hill in each row) than in T5 (the **blue** hill in each row). The difference in parameter sensitivity becomes more conspicuous when the models are trained for more steps. These observations suggest that pretraining with model-generated signals makes the sensitivity of parameters more concentrated. In other words, the amount of each parameter's contribution becomes more balanced with METRO-style pretraining, which leads to a more sufficiently trained model.

7 Conclusion

This paper presents a new method for improving the zero-shot generalization of T5-like text-to-text Transformers by incorporating model-generated signals in the pretraining process. METRO-T0, the model sufficiently trained using our redesigned pretraining method, is highly parameter efficient and compute efficient. We hope that the success of our approach could inspire further work on efficient big LM pretraining and prompt-based learning.

Limitations

This work focuses on pretraining large language models for zero-shot generalization. Although our proposed method is more efficient than baselines, it still requires significant computational resources, specifically GPU resources. The GPU resources used and training time are detailed in Appendix A.6. Our study is also limited by the computational budget, preventing us from training models as large as GPT-3 or $\text{T0}_{11\text{B}}$. However, our *large++* model (775M parameters) already rivals or outperforms previous state-of-the-art models.

Ethics Statement

This work proposes and releases language models that are pretrained on web-crawled data and finetuned on a large collection of NLP datasets. These models may perpetuate social stereotypes and disparities reflected in the training data, or accidentally reveal private information. Mitigating these risks presents a significant open research challenge that calls for collective efforts within the NLP community. Therefore, it is recommended to take appropriate measures to assess risks and potential harms in the application context before deployment.

Acknowledgement

Linyuan Gong and Alvin Cheung are partially supported by the National Science Foundation through grants IIS-1955488, IIS-2027575, CCF-1723352, ARO W911NF2110339. We thank Mingrui Shen for his support providing computing infrastructure support on the finetuning work flow, Guolin Ke for his support in establishing the data processing pipelines for our pretraining corpora, and anonymous reviewers for their constructive feedback.

References

Stephen H. Bach, Victor Sanh, Zheng-Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesht Sharma, Taewoon Kim, M Saiful Bari, Thibault Fevry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged S. Al-shaibani, Shanya Sharma, Urmish

Thakker, Khalid Almubarak, Xiangru Tang, Xiangru Tang, Mike Tian-Jian Jiang, and Alexander M. Rush. 2022. Promptsource: An integrated development environment and repository for natural language prompts.

Payal Bajaj, Chenyan Xiong, Guolin Ke, Xiaodong Liu, Di He, Saurabh Tiwary, Tie-Yan Liu, Paul Bennett, Xia Song, and Jianfeng Gao. 2022. Metro: Efficient denoising pretraining of large scale autoencoding language models with model generated signals. *arXiv preprint arXiv:2204.06644*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Zewen Chi, Shaohan Huang, Li Dong, Shuming Ma, Saksham Singhal, Payal Bajaj, Xia Song, and Furu Wei. 2021. Xlm-e: Cross-lingual language model pre-training via electra. *arXiv preprint arXiv:2106.16138*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Zhao, Yanping Huang, Andrew Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT 2019*, pages 4171–4186.

Yuxin Fang, Li Dong, Hangbo Bao, Xinggang Wang, and Furu Wei. 2022. Corrupted image modeling for self-supervised visual pre-training. *arXiv preprint arXiv:2202.03382*.

Suyu Ge, Chenyan Xiong, Corby Rosset, Arnold Overwijk, Jiawei Han, and Paul Bennett. 2023. Augmenting zero-shot dense retrievers with plug-in mixture-of-memories. *arXiv preprint arXiv.2302.03754*.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *CoRR*, abs/2009.03300.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710.

Chen Liang, Haoming Jiang, Simiao Zuo, Pengcheng He, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Tuo Zhao. 2022. No parameters left behind: Sensitivity guided adaptive learning rate for training large transformer models. *CoRR*, abs/2202.02664.

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

Shengjie Luo, Shanda Li, Shuxin Zheng, Tie-Yan Liu, Liwei Wang, and Di He. 2022. Your transformer may not be as powerful as you expect. *arXiv preprint arXiv:2205.13401*.

Yu Meng, Chenyan Xiong, Payal Bajaj, Saurabh Tiwary, Paul Bennett, Jiawei Han, and Xia Song. 2021. COCO-LM: Correcting and contrasting text sequences for language model pretraining. In *Proceedings of NeurIPS 2021*.

Yu Meng, Chenyan Xiong, Payal Bajaj, Saurabh Tiwary, Paul Bennett, Jiawei Han, and Xia Song. 2022. Pretraining text encoders with adversarial mixture of training signal generators. *arXiv preprint arXiv:2204.03243*.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268.

Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B. Hall, Daniel Cer, and Yinfei Yang. 2021. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models. *CoRR*, abs/2108.08877.

Adam Polyak and Lior Wolf. 2015. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*.

Victor Sanh, Albert Webson, Colin Raffel, Stephen Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Fevry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M Rush. 2022. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*.

Trieu H Trinh and Quoc V Le. 2018. A simple method for commonsense reasoning. *arXiv preprint arXiv:1806.02847*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *EMNLP Workshop BlackboxNLP*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Zhenhui Xu, Linyuan Gong, Guolin Ke, Di He, Shuxin Zheng, Liwei Wang, Jiang Bian, and Tie-Yan Liu. 2020. MC-BERT: efficient language pre-training via a meta controller. *CoRR*, abs/2006.05744.

A Appendix

A.1 Pretraining Corpus

Our *base* model is pretrained on English Wikipedia and BookCorpus (16GB of texts). We encode the pretraining corpus with an uncased vocabulary of 32,768 BPE tokens. This setup is similar to vanilla BERT (Devlin et al., 2019).

Our *base++/large++* model is pretrained on a mixed corpus of 160GB texts, which consists of English Wikipedia, BookCorpus, OpenWebText (Gokaslan and Cohen, 2019), CC-News (Liu et al., 2019), and STORIES (Trinh and Le, 2018). We encode the corpus with a cased vocabulary of 64,000 BPE tokens. This setup is similar to RoBERTa (Liu et al., 2019), COCO-LM (Meng et al., 2021), and METRO-LM (Bajaj et al., 2022).

As a reference, T0 (Sanh et al., 2022) models and Flan-T5 (Chung et al., 2022) are all based on the original T5 model by Raffel et al. (2019). The pretraining corpus is the C4 corpus (Raffel et al., 2019) of 800GB of texts based on CommonCrawl. They encode the corpus with a cased vocabulary of 32k BPE tokens.

A.2 Pretraining Hyperparameters

The hyperparameters we used to pretrain METRO-T0 and our T0 baseline are listed in Table 5.

Hyperparameters	**Base**	**Base++**	**Large++**
Encoder Layers	12	12	24
Decoder Layers	12	12	24
Auxiliary Layers*	4	4	6
Hidden Dimension	768	768	1,024
Peak Learning Rate	4e-4	2e-4	2e-4
Batch Size	2,048	2,048	2,048
Warm-Up Steps	10,000	10,000	10,000
Total Steps	125,000	2,000,000	1,335,000
Sequence Length	512	512	512
Relative Position Encoding Buckets	32	32	64
Relative Position Encoding Max Distance	128	128	128
Loss multipliers (λ_{MLM}, λ_{RTD}, λ_{CLM})*	(1, 50, 1)	(1, 50, 1)	(1, 50, 1)
Adam ϵ	1e-6	1e-6	1e-6
Adam (β_1, β_2)	(0.9, 0.98)	(0.9, 0.98)	(0.9, 0.98)
Clip Norm	-	2.0	2.0
Dropout	0.1	0.1	0.1
Weight Decay	0.01	0.01	0.01

Table 5: Pretraining hyperparameters for METRO-T0 and our T0 baselines. Rows with an "*" are specific to METRO-style pretraining and not applicable to our T0 baselines. We only train our *large++* model for 1.3M steps (instead of 2M steps) due to limitations of computational budgets but it still yields impressive performance.

In pretraining, we use 15% masking ratio for the auxiliary MLM pretraining task. We create a `[MASK]` symbol for each masked token. Each token in X^{noise} is sampled from the softmax distribution predicted by the auxiliary model for each `[MASK]` symbol. The weight of each pretraining objective is $\lambda_{\text{MLM}} = 1$, $\lambda_{\text{RTD}} = 50$, and $\lambda_{\text{CLM}} = 1$, following Meng et al. (2021). In both the auxiliary transformer and the main transformer, we use shared token embeddings in the embedding layer and the language modeling head.

We have three projection heads in our model: MLM head on the auxiliary transformer, RTD head on the main transformer's encoder, and CLM head on the main transformer's decoder. Both the MLM and CLM head are a single linear transformation. We use RoBERTa-style projection head for the RTD head, which contains a linear projection, a ReLU activation, a layer norm and another linear projection. For the

RTD on decoder (complex CLM head) ablation, we use a RoBERTa-style head as the architecture of the CLM head.

A.3 Data for Prompt-Finetuning

Following Sanh et al. (2022), we finetune our models on three training mixtures, *T0 Train* (39 datasets), *T0+ Train* (49 datasets), and *T0++ Train* (55 datasets), respectively. Each dataset is associated with multiple (8.03 on average) prompt templates that are used to format example instances to input and target pairs. Please refer to Sanh et al. (2022) for more details about our finetuning datasets.

A.4 Prompt-Finetuning Hyperparameters

Once we have METRO-T5 pretrained on a natural language corpus, we discard the auxiliary model and keep the main model, which is a standard text-to-text Transformer. We finetune this model on multi-task training mixtures of NL-prompted datasets proposed by Sanh et al. (2022). Once the model parameters are initialized with pretrained METRO-T5, the finetuning procedure is standard sequence-to-sequence learning: the input sequence is fed to the encoder, and the target sequence serves as the ground truth to compute the cross-entropy loss of the decoder output. Each model is finetuned using hyperparameters listed in Table 6. Basically, we use the same hyperparameters as pretraining, except the peak learning rate is reduced to 0.1x and each target sequence is truncated to a max length of 256. We do not perform any checkpoint selection or hyperparameter selection, and simply use the last checkpoint at 125k steps of this single run for evaluation.

Hyperparameters	**Base**	**Base++**	**Large++**
Peak Learning Rate	4e-5	2e-5	2e-5
Total Steps	125,000	125,000	125,000
Source Sequence Length	512	512	512
Target Sequence Length	256	256	256
Clip Norm	-	-	-

Table 6: Hyperparameters for prompt-finetuning METRO-T5 and our pretrained T5 baseline. All hyperparameters not mentioned in this table is the same as in the pretraining procedure.

A.5 Evaluation

We evaluate zero-shot generalization on the *T0 Eval* benchmark (Sanh et al., 2022) and the *Massive Multi-task Language Understanding* (*MMLU*) benchmark (Hendrycks et al., 2020). T0 Eval consists of 11 held-out datasets in natural language inference, coreference, word sense disambiguation, and sentence completion, and details are shown in Table 7 MMLU includes example questions from 57 tasks such as maths, history, law, and medicine. Please refer to Hendrycks et al. (2020) for more details about MMLU.

	Size	**Task**	**Metric**
RTE	277	Natural language inference	Accuracy
CB	56	Natural language inference	Accuracy
ANLI	3,200	Natural language inference	Accuracy
WSC	104	Coreference resolution	Accuracy
Winogrande XL	1,267	Coreference resolution	Accuracy
COPA	100	Sentence completion	Accuracy
StoryCloze 2016	1,871	Sentence completion	Accuracy
HellaSwag	10,042	Sentence completion	Accuracy
WiC	638	Word Sense Disambiguation	Accuracy

Table 7: The overview of the *T0 Eval* benchmark for prompt learning.

Each task in *T0 Eval* or *MMLU* is formulated as multiple-choice questions. We compute the log probability of each choice under the finetuned model and select the choice with the highest log probability as the prediction.

A.6 Implementation Details

Implementation We implement our T0 baseline and METRO-T0 based on `fairseq`[1]. The prompt templates to format the finetuing data are from the `promptsource`[2] library (Bach et al., 2022). We evaluate pretrained models on the *T0 Eval* benchmark using `transformers`[3] and `t-zero`[4].

Pretraining and Finetuning Costs. Pretraining METRO-T5 in the *base* setting takes 20.8 hours on 64x NVIDIA A100 (40GB) GPUs. The pretraining cost of METRO-T5 is T5 (our implementation) plus the auxiliary transformer, whose number of layers is 1/3 of the main transformer's encoder. Pretraining METRO-T5 in the *base*++ setting takes 159 hours on 128x NVIDIA A100 (40GB) GPUs. Pretraining METRO-T5 in the *large*++ setting takes 289 hours on 256x NVIDIA A100 (40GB) GPUs. In finetuning, we remove the auxiliary transformer and the RTD and CLM heads, so the finetuning cost of METRO-T5 and T5 are the same. Prompt-finetuning each *base/base*++ model takes about 22 hours on 64x NVIDIA V100 (16GB) GPUs. Prompt-finetuning each *large*++ model takes about 70 hours on 64x NVIDIA V100 (16GB) GPUs.

A.7 Full Results on *T0 Eval*

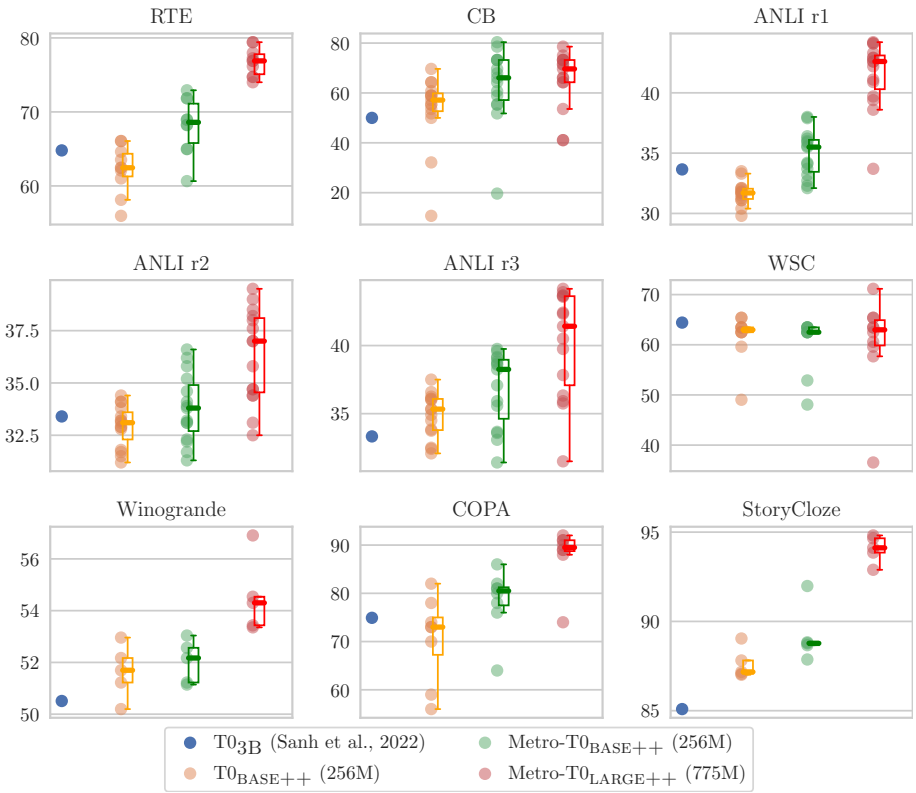


Figure 8: Prompt-based learning results of METRO-T0 versus our T0 baseline and T0$_{3B}$ by Sanh et al. (2022) on all 9 tasks in the *T0 Eval* benchmark. Each point denotes the accuracy using one prompt template, except that the median accuracy over all templates of T0$_{3B}$ is indicated by the **blue** point.

Figure 8 results of METRO-T0 versus our T0 baseline and T0$_{3B}$ by Sanh et al. (2022) on all 9 tasks in the *T0 Eval* benchmark. The results shows that METRO-T0$_{LARGE++}$, having only 775M parameters,

[1] https://github.com/facebookresearch/fairseq
[2] https://github.com/bigscience-workshop/promptsource
[3] https://huggingface.co/docs/transformers/index
[4] https://github.com/bigscience-workshop/t-zero

consistently outperforms $T0_{3B}$ over all tasks on the *T0 Eval* benchmark.

A.8 Evaluation on MMLU

The prompt template used to evaluate our models MMLU is the prompt template from the *AI2 Reasoning Challenge (AI2-ARC)* concatenated with 5 passages in MS MARCO (Nguyen et al., 2016). These 5 passages are selected via *dense retrival* using T5-ANCE (Ge et al., 2023; Ni et al., 2021), which maps a query to a single vector to retrieve similar passage from the corpus. Adding densely-retrieved passages to prompts is a standard approach to enhance LM's performance on zero-shot prompting. This approach is named *retrieval augmentation*. All T0 and METRO-T0 results reported in Table 3 are evaluated using this prompt template with retrieval augmentation.

On the other hand, all Flan-T5 results reported in Table 3 are numbers reported in their paper. For each model, we take the maximum score of the reported "direct" prompting performance and the "chain-of-thought (CoT)" prompting performance. Both prompt templates are not publicly available as of the time this paper is written.

As a result, Table 3 involves comparisons across multiple prompt templates. So in Table 8, we present the performance of each model using the *plain* AI2-ARC prompt template *without* retrieval augmentation or CoT.

Model	Params	MMLU
AI2-ARC Prompt Template		
$T0{+}{+}_{BASE}$	226M	31.5
METRO-$T0{+}{+}_{BASE}$	226M	**31.9**
Flan-$T5_{BASE}$ (Wei et al., 2022)	223M	33.8
$T0{+}{+}_{BASE++}$	256M	37.8
METRO-$T0{+}{+}_{BASE++}$	256M	**38.9**
Flan-$T5_{LARGE}$ (Wei et al., 2022)	750M	39.0
$T0{+}{+}_{11B}$ (Sanh et al., 2022)	11B	30.9
METRO-$T0{+}{+}_{LARGE++}$	775M	**43.4**
AI2-ARC Prompt Template + Retrieval Augmentation		
$T0{+}{+}_{BASE}$	226M	37.5
METRO-$T0{+}{+}_{BASE}$	226M	**38.3**
Flan-$T5_{BASE}$ (Wei et al., 2022)	223M	40.4
$T0{+}{+}_{BASE++}$	256M	41.7
METRO-$T0{+}{+}_{BASE++}$	256M	**42.7**
Flan-$T5_{LARGE}$ (Wei et al., 2022)	750M	41.4
$T0{+}{+}_{11B}$ (Sanh et al., 2022)	11B	35.6
METRO-$T0{+}{+}_{LARGE++}$	775M	**48.0**
Reported numbers by Chung et al. (2022)		
Flan-$T5_{BASE}$ (Wei et al., 2022)	223M	35.9
GPT-3_{175B} (Brown et al., 2020)	175B	43.9
Flan-$T5_{LARGE}$ (Wei et al., 2022)	750M	45.1

Table 8: Full prompt learning results on the *MMLU* dataset in three setups. All reported results use accuracy averaged over 57 subtasks as their metric.

The result in Table 8 shows that METRO-T0++ still consistently outperforms the T0 baseline and similar-sized Flan-T5 models when they are evaluated using the same prompt template.

A.9 Example of the Challenge of Ill-Formed Target

In our discussion about "decoding target" inSection 4, we claim that *"masked tokens only"* is an *ill-formed target* for the CLM objective in METRO-style pretraining of T5. This section shows a concrete example where such ill-formed target leads to ambiguities.

In Table 9, the original sentence is "`1 2 3 4 5`". Using different random samples of masked positions, we can derive two masked sequences as the input of the auxiliary model: "`1 M M M 5`" and "`1 2 M M 5`".

Table 9: An example where ill-formed target leads to ambiguities. Each number denotes a distinct subword token. M denotes the special token [MASK]. In "Auxiliary Model Prediction", a token shown in green denotes a correct prediction, where a token shown in red denotes a wrong prediction.

Sentence	1	2	3	4	5
Auxiliary Model Input 1	1	M	M	M	5
Auxiliary Model Prediction		2	6	4	
Main Model Input	1	2	6	4	5
Main Model Target	2	3	4		
Auxiliary Model Input 2	1	2	M	M	5
Auxiliary Model Prediction			6	4	
Main Model Input	1	2	6	4	5
Main Model Target	3	4			

The difference is whether "2" is masked or not. So the target for the decoder corrective LM objective will be "2 3 4" and "3 4" respectively. After we have the masked input, the auxiliary model, which is a *masked language model (MLM)*, tries to fill masked positions with predicted tokens "2 6 4" and "6 4" respectively. The resulting main model input is "1 2 6 4 5" for both cases, but the target is "2 3 4" for case 1 and "3 4" for case 2. This is an ambiguity where the main model is unsure where it should begin to generate predictions: "2" or "3".

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
We discussed the limitation of our work in a separate section after the main paper

☑ A2. Did you discuss any potential risks of your work?
We discussed the limitation of our work in a separate section after the main paper

☑ A3. Do the abstract and introduction summarize the paper's main claims?
We summarize the paper's main claims in the last two paragraphs of the introduction

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

We listed all scientific artifacts we used in Appendix A and Appendix F

☑ B1. Did you cite the creators of artifacts you used?
We cited all scientific artifacts we used in Appendix A and Appendix F

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
We discussed the compliance of our usage in Appendix A and Appendix F

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Not applicable. All artifacts we use allows usage in open academic research

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
We follow previous works to use unanonymized web-crawed corpora for large-scale pretraining. To our best knowledge, there is no anonymized corpus for pretraining LMs that is as large as the one that we and our baselines are using. Therefore, using this unanonymized pretraining corpus is the only way we can conduct fair-comparisons against baselines.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
The documentations can be found in Appendix A and Appendix F

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
The statistics can be found in Appendix A, Appendix C, and Appendix E

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

C ☑ Did you run computational experiments?

Left blank.

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
In Appendix F

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
In Appendix B, Appendix D, and Appendix F

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Figure 1 and Figure 8 shows the boxplot of results as well as one point for each run to describe the distribution of accuracies over multiple runs

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
In Appendix A and Appendix F

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
Not applicable. Left blank.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
Not applicable. Left blank.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
Not applicable. Left blank.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
Not applicable. Left blank.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
Not applicable. Left blank.