Insights into Pre-training via Simpler Synthetic Tasks

Yuhuai Wu[12*]
yuhuai@cs.stanford.edu

Felix Li[3*]
fzli@berkeley.edu

Percy Liang[1]
pliang@cs.stanford.edu

[1]Stanford University
[2]Google Research
[3]UC Berkeley

Abstract

Pre-training produces representations that are effective for a wide range of downstream tasks, but it is still unclear what properties of pre-training are necessary for effective gains. Notably, recent work shows that even pre-training on synthetic tasks can achieve significant gains in downstream tasks. In this work, we perform three experiments that iteratively simplify pre-training and show that the simplifications still retain much of its gains. First, building on prior work, we perform a systematic evaluation of three existing synthetic pre-training methods on six downstream tasks. We find the best synthetic pre-training method, `LIME`, attains an average of 67% of the benefits of natural pre-training. Second, to our surprise, we find that pre-training on a simple and generic synthetic task defined by the `Set` function achieves 65% of the benefits, almost matching `LIME`. Third, we find that 39% of the benefits can be attained by using merely the parameter statistics of synthetic pre-training. We release the source code at https://github.com/felixzli/synthetic_pretraining.

1 Introduction

Pre-training on a large amount of data from sources such as text from the web—*natural pre-training*—is effective for a wide range of downstream tasks (Devlin et al., 2019; Brown et al., 2020; Chen et al., 2021; Lu et al., 2021). More surprisingly, recent works (Wu et al., 2021; Krishna et al., 2021; Ri & Tsuruoka, 2022; Anderson & Farrell, 2022) show that pre-training on data that is fully synthetically generated—*synthetic pre-training*—can also provide substantial gains, partially closing the gap between training from a randomly initialized model and a naturally pre-trained model.

What properties of pre-training are necessary for effective gains? In this paper, we conduct a careful empirical study to help understand this question. We perform three experiments that iteratively simplify natural pre-training, while still retaining much of the benefits of natural pre-training. See Figure 1 for an overview of the results obtained in our three experiments.

First, we perform a systematic evaluation of three previously proposed synthetic pre-training methods over six downstream tasks. Prior works only evaluated each synthetic pre-training method on a single downstream domain: Wu et al. (2021) on mathematical reasoning benchmarks, Chiang & Lee (2021) on GLUE, Krishna et al. (2021) on summarization, and Ri & Tsuruoka (2022) on language modeling and dependency parsing. We instead show how general these methods are as well as how different synthetic pre-training methods compare against each other. We find that the best performing synthetic task, `LIME` (Wu et al., 2021), closes a significant fraction of the gap between random initialization and natural pre-training for summarization (85%), semantic parsing (83%), reading comprehension (55%), code translation (43%), and retrosynthesis (49%).

[*]Equal Contribution.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

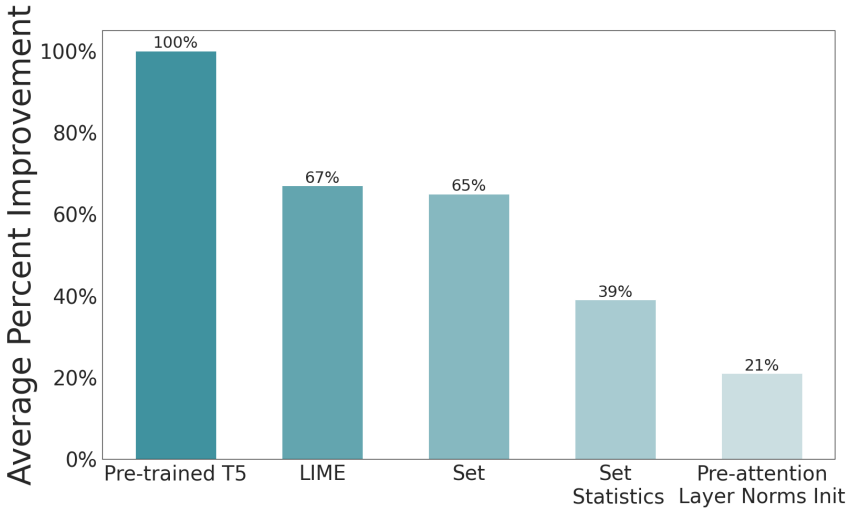

Figure 1: We compare downstream task performance between natural language pre-training (`Pre-trained_T5`), synthetic pre-training (`LIME`), simpler synthetic pre-training (`Set`), initializing from the statistics of Set pre-training (`Set` Statistics), and initializing just the pre-attention layer norms to a lower value (Pre-attention Layer Norm Init). The average percent improvement is computed by taking the average over our six downstream tasks of $\frac{\text{Performance(Initialization)} - \text{Performance(Random_Init)}}{\text{Performance(Pre-trained_T5)} - \text{Performance(Random_Init)}}$.

Second, we discover a surprisingly simple and generic synthetic task that captured nearly the same effectiveness provided by the existing best synthetic task `LIME`, closing an average of 65% of the gap to natural pre-training (compared to 67% for `LIME`). This simple task, which we call `Set`, consists of outputting the input sequence without duplicate tokens in its original order, for example:

Input: a a c b a b $\Rightarrow$ Output: a c b.

Third, to simplify the initialization even more, we ask whether only the statistics of the pre-trained weights can explain the effectiveness of synthetic pre-training. We find that initializing parameters with statistics of a synthetic pre-trained model provides large benefits for some downstream tasks, closing 39% of the gap to natural pre-training. Simplifying further allows us to see that by initializing all pre-attention layer norm parameters to a lower value (just one number!), we close more than 40% of the gap to natural pre-training performance for summarization and semantic parsing, and an average of 21% over all six tasks.

2 Experimental Setup

Architecture We trained a T5-Small model, which is a 60 million parameter encoder-decoder transformer model with 6 encoder layers, 6 decoder layers, 8 heads, head dimension 64, and MLP dimension 2048 (Raffel et al., 2020).

Training Details For synthetic pre-training, we use the same hyperparameters that the off-the-shelf language pre-trained T5-small was trained with: AdaFactor optimizer, batch size 128, sequence length 512, and inverse square root learning rate $1/\sqrt{\max(n, 10000)}$ where n is the current training step. We evaluate token validation accuracy every 5000 training steps. For all synthetic tasks besides `Dyck`, we fine-tuned with the first checkpoint that the model reaches above 99% token validation accuracy. For `Dyck`, the model's validation accuracy plateaued without reaching above 99%, so we chose to fine-tune using the checkpoint with max validation accuracy of 77.7%. We provide further pre-training details in Appendix A and fine-tuning details in Appendix B.

Downstream Tasks We fine-tuned synthetically pre-trained models on a diverse suite of downstream tasks: 1) Java to C# code translation (10K training examples) (Lu et al., 2021); 2) two semantic parsing benchmarks, MTOP (17K training examples) (Li et al., 2021) and WebQSP (2.7K training

Table 1: Evaluation of three previously proposed synthetic pre-training tasks and two simpler synthetic pre-training tasks. For tasks where we report both Exact Match (EM) and F1, the Average column is computed using EM.

	CNNDM-10K ROUGE1	MTOP EM/F1	WebQSP EM/F1	SQuAD EM/F1	Code Trans. EM	Retrosyn. EM	Average	
				Baselines				
`Pre-trained_T5`	**35.8**[1]	**81.0/95.2**	**83.1/91.9**	**77.5/86.0**	**61.6**	**43.1**	**63.7**	**100%**
`Random_Init`	18.9	38.6/83.0	28.2/72.9	17.2/25.2	57.0	39.2	33.2	**0%**
				Section 3: Benchmarking Existing Synthetic Pre-training				
`LIME`	**33.2**	**73.7/94.0**	**75.2/88.7**	**50.4/62.1**	**59.0**	**41.1**	**55.4**	**67%**
`Dyck`	27.1	65.9/91.9	58.5/83.5	50.3/62.4	58.8	40.4	50.2	49%
`Nons_Summary`[2]	32.0	68.0/92.7	65.2/85.2	48.4/60.1	57.3	39.6	51.8	47%
				Section 4: Simpler Synthetic Pre-Training				
`Set`	**32.8**	**71.7/93.7**	**72.2/88.2**	**48.3/60.0**	**59.4**	**40.9**	**54.2**	**65%**
`Identity`	30.2	68.6/93.0	69.8/86.8	26.1/35.4	57.8	40.5	48.8	46%

examples) (Yih et al., 2016), consisting of converting a natural language query to a logical form; 3) USPTO-50K retrosynthesis (40K training examples) (Liu et al., 2017), a task that consists of predicting possible reactants when given a product as input; 4) the reading comprehension benchmark SQuAD 1.1 (87K training examples) (Rajpurkar et al., 2016); and 5) the summarization benchmark CNNDM-10K[3] which is 10K training examples from the CNNDM (Krishna et al., 2021). See full descriptions and input-output examples for every downstream task in Appendix D.

Baselines We compared against randomly initialized (`Random_Init`) and off-the-shelf natural language pre-trained T5-Small (`Pre-trained_T5`). `Pre-trained_T5` was trained for 524K steps on the C4 dataset with the span-corruption objective (Raffel et al., 2020). We also compared against T5-Small trained on Wikipedia (Guo et al., 2020) for 10K steps (`Wiki_10K`) with the same pre-training objective as `Pre-trained_T5`.

Source code We release the source code to reproduce the experiments at `https://github.com/felixzli/synthetic_pretraining`.

3 Benchmarking Existing Synthetic Pre-training

We benchmark three previously proposed synthetic pre-training tasks across six downstream tasks. We compute how much of the benefits using natural pre-training are gained by each synthetic pre-training method by taking the average over six downstream tasks of $\frac{\text{Performance(Initialization)} - \text{Performance(Random_Init)}}{\text{Performance(Pre-trained_T5)} - \text{Performance(Random_Init)}}$. We show these numbers in the right-most column of Table 1. We find that `LIME` achieves the best performance, gaining 67% of the benefits using natural pre-training.

3.1 Synthetic Tasks

For each synthetic task, we generated one million examples. To generate synthetic task data, we sampled tokens a vocabulary of size 32K, matching the vocabulary size of `Pre-trained_T5`. Details about data generation can be found in the original papers. We provide an example of each task in Figure 2.

[1] This number was obtained using pre-trained `T5_1.1` instead of `T5_1.0` which we used for all other tasks. Using pre-trained `T5_1.0` would be an unfair comparison because it has already been trained on all 290K CNNDM task training data. Full differences between `T5_1.1` and `T5_1.0` are described in Appendix Section C.

[2] We obtained different results than Krishna et al. (2021) on CNNDM-10K because of our choice of optimizer. We were able to match their results when we used Adam like they did in their work instead of Adafactor. We chose to use Adafactor instead of Adam because training took about three times less steps.

[3] We followed Krishna et al. (2021) to evaluate on CNNDM-10K instead of the full CNNDM. They used CNNDM-10K because less training examples makes the impact of pre-training more visible.

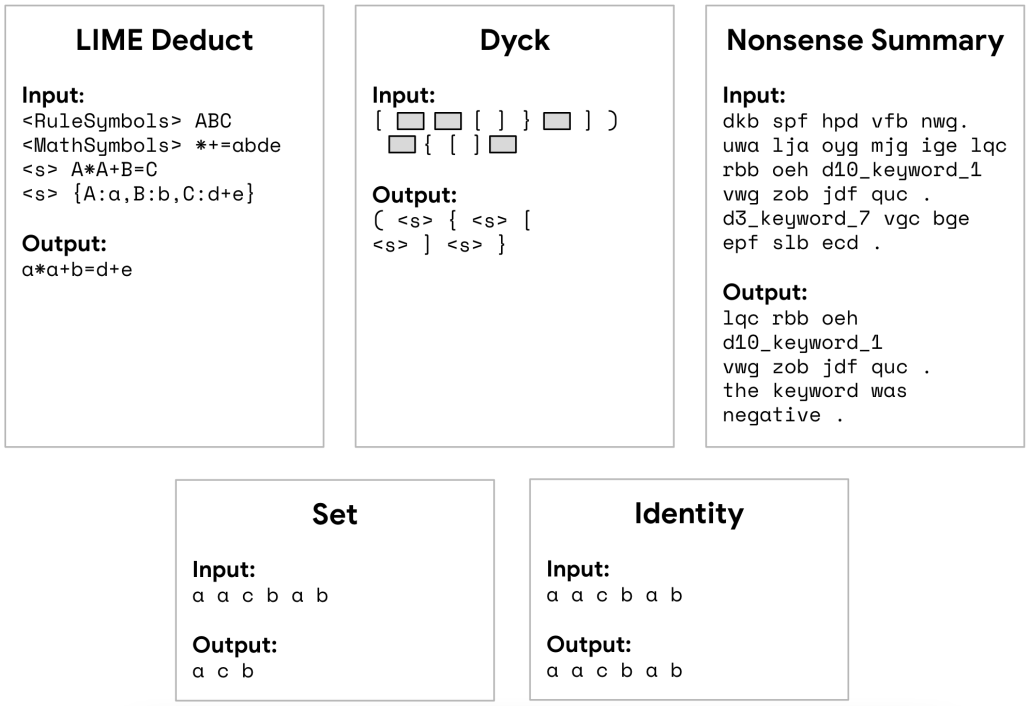

Figure 2: We show examples of Section 3 previously proposed synthetic tasks in the top row and Section 4 simpler synthetic tasks in the bottom row. **Top Left:** `LIME` deduct task, whose input consists of a "rule string" and "substitution dictionary" and output is a "result string." The "rule symbols" and "math symbols" are written as actual letters and math operations for reader clarity. In real generated data, each example's rule and math symbols are randomly sampled tokens. **Top Middle:** T5 style masked language modelling with `Dyck` language. In the input, the grey boxes indicate masked out spans. The model is trained to predict the masked spans, each separated by a special token `<s>`. In this example, the "noise_density" is 0.33 (so 5 out of 15 input brackets are corrupted) and the "span_corruption_length" is 1 (so each masked span is of length 1). In actual generated data, the "span_corruption_length" is 3 and the "noise_density" is 0.15, which are the same parameter values used for natural language `Pre-trained_T5`. **Top Right:** `Nonsense_Summary` task. In this example, two operations to create the summary are "copy sentence containing a keyword" and "identify keyword sentiment." **Bottom Left:** `Set` task. **Bottom Right:** `Identity` task.

LIME `LIME` is a set of three tasks—Deduct, Induct, and Abduct—inspired by Charles Peirce's three reasoning primitives (Wu et al., 2021). Each task consists of three elements: a rule string, a dictionary that represents substitutions, and a result string that is the result of applying those substitutions to the rule string. The three tasks are then constructed by using two of the three elements as inputs to predict the remaining element.

Dyck Artificial Language We consider a Dyck language with multiple bracket types (taken from Ri & Tsuruoka (2022)), which we refer as `Dyck`. We turn generated text into a sequence-to-sequence task by using the T5 span-corruption pre-training objective (Raffel et al., 2020). The motivation behind the design of `Dyck` is how "sentences of natural language often have dependency relations where the existence of a certain word can be predictive of another word" (Ri & Tsuruoka, 2022).

Nonsense Summary Krishna et al. (2021) developed the nonsense summarization task, which we refer to as `Nonsense_Summary`, by conducting a qualitative analysis of summarization data and identifying 21 elementary operations used to create summaries such as "CopyFirstSentence" and "CopyQuoted." Each synthetic task example consists of an input that is a nonsense document made of randomly sampled tokens and an output that is the corresponding nonsense summary consisting of applying a random subset of the 21 operations to the document.

3.2 Results

Table 1 shows results of evaluating the synthetic tasks across six downstream tasks, and Appendix Table 4 shows results of ablating the dataset size for retrosynthesis, CNNDM, and SQuAD.

Synthetic pre-training provides large gains over random initialization. We observed large gains from all three synthetic tasks we evaluated. Out of these, LIME provided the most benefits across all six downstream tasks. We computed how much of the natural pre-training benefit is captured by LIME as $\frac{\text{Performance(LIME)} - \text{Performance(Random_Init)}}{\text{Performance(Pre-trained_T5)} - \text{Performance(Random_Init)}}$. LIME closed 84.6%, 82.8%, 85.6%, 55.1%, 43.5%, and 48.7% of the gap for CNNDM-10K, MTOP, WebQSP, SQuAD, Code Translation, and Retrosynthesis respectively.

Task or domain prior is not required for pre-training benefits. For example, LIME benefits tasks including summarization (CNNDM-10K) and reading comprehension (SQuAD) despite LIME pre-training data involving no natural language like Pre-trained_T5, representing no structure of natural language like Dyck, and not being explicitly designed to reflect the operations used in summarization like Nonsense_Summary.

Varying degrees of benefits. Comparing Nonsense_Summary and Dyck, we see that the two synthetic tasks benefit MTOP, WebQSP, and SQuAD similarly, yet their benefits for the three other tasks are very different. The difference between Nonsense_Summary and Dyck as well as the superiority of LIME are examples of how different synthetic pre-training can have significantly different effectiveness.

Synthetic pre-training is not more efficient. One of the advantages argued for synthetic pre-training is its computational efficiency (Wu et al., 2021; Krishna et al., 2021). To explore this potential advantage, we pre-trained our model using the T5 span-corruption pre-training objective on Wikipedia (Guo et al., 2020) for 10K steps, and the result is shown in Table 1 labelled Wiki_10K. Wikipedia pre-training for 10K steps provided more significant gains than LIME pre-training for 30K steps and Set pre-training for 10K steps. Based on this result, we see no evidence that the evaluated synthetic pre-training tasks provides computational efficiency benefits over natural language pre-training, unlike what Wu et al. (2021) suggested.

Natural language pre-training benefits non-natural language tasks. Natural language pre-training, including Wiki_10K, outperforms synthetic pre-training on the non-natural language tasks of Code Translation and Retrosynthesis.

4 Simpler Synthetic Pre-training

Encouraged by the results in the previous section, we want to explore what is necessary for synthetic pre-training benefits. We evaluate pre-training with simpler and more generic synthetic tasks similar to basic Python API functions. Surprisingly, a simple and generic task defined by the Set function provides significant benefits (65%), almost matching the previous best synthetic task LIME (67%). Furthermore, another simple task defined by the Identity function also provided significant benefits (46%).

4.1 Simpler Synthetic Tasks

We constructed 18 simpler synthetic tasks similar to basic Python API functions. As we did for the three existing synthetic tasks, for each simpler synthetic task we generated one million data points with a vocabulary of 32K tokens, matching the vocabulary size of Pre-trained_T5. In Appendix G.1, we define each task, explain how examples are randomly generated, and provide input-output examples. We first evaluated all tasks on CNNDM-10K. We then evaluated Set, one of the best performing tasks on CNNDM-10K, and Identity, one of the simplest tasks, on all six downstream tasks.

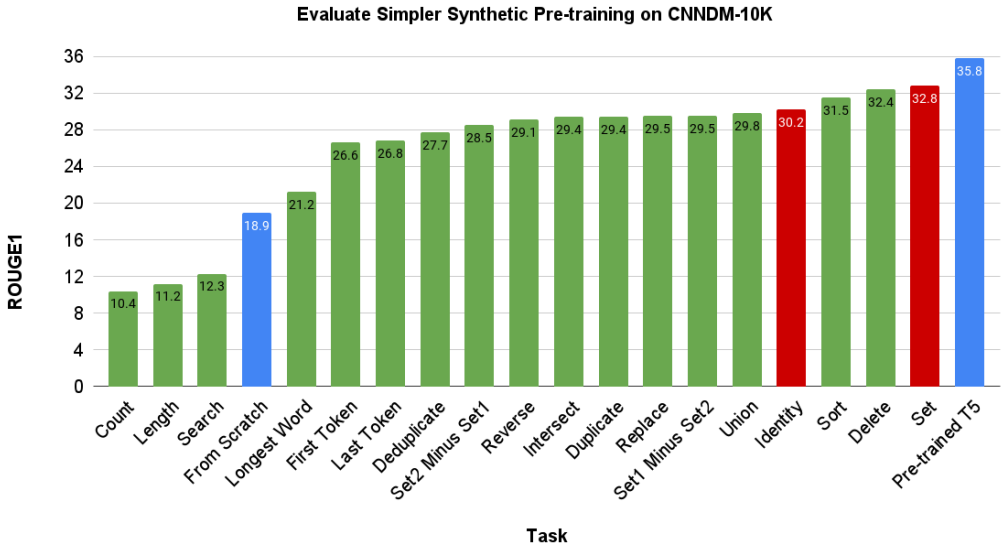

Figure 3: For each simple synthetic task, we pre-train on it and then fine-tune on CNNDM-10K. `Set` performed the best and `Identity`, one of the simplest tasks, provided significant benefit. The `Set` and `Identity` tasks are defined in 4.1, and the other tasks are defined in G.

Set The `Set` task consists of outputting the input sequence in its original order except without tokens appearing more than once. An example input is [a b b a a a c d c], and the corresponding output is [a b c d]. The `Set` task data generation is explained in Appendix G.

Identity The `Identity` task consists of copying the input tokens to the output. An example input is [a d e a a], and the corresponding output is [a d e a a]. To generate one example, we uniformly sample a sequence of tokens between length 10 and length 220 to be the input and output.

4.2 Results: simpler synthetic tasks provide similar benefits as existing synthetic tasks

Figure 3 shows results on CNNDM-10K for models pre-trained on each of 18 simpler synthetic tasks. Table 1 shows results on all six downstream tasks for `Set` and `Identity` pre-trained models. Averaged across the six downstream tasks, `Set` and `Identity` gain 65% and 46% of the natural pre-training benefits compared to 67% for `LIME`, the best previously proposed synthetic task evaluated in Section 3. `Identity` provides at least as much benefit as Nested Language and Nonsense Summary for all tasks besides Code Translation, for which Nested Language benefits more.

The significant benefits of `Set` and `Identity` pre-training suggest that the complexities of the three previously proposed synthetic tasks we evaluated—representing three fundamental logical operations, having a structural property that mimics natural language, and reflecting operations used for summarization—may not be the necessary factors for why pre-training on those tasks provides benefits. Also, due to their simplicity, `Set` and `Identity` may be useful for future work towards understanding pre-training and how to design better synthetic tasks.

5 Even Simpler Synthetic Pre-training with Statistics

Inspired by the previous section, we tried to find even simpler initializations that achieve the the benefits of pre-training. We explored how benefits of synthetic pre-training can be attained by initialization based only on statistics of pre-trained weights. We found: (1) Initialization using the mean and standard deviation (SD) of synthetic pre-trained parameters can gain more than 70% of the benefit from synthetic pre-training for CNNDM-10K and MTOP (or an average of 39% of natural pre-training benefits). (2) Initialization using simpler schemes involving less statistics can still provide benefits. (3) Specifically the pre-attention layer norm plays an important role for initialization benefits.

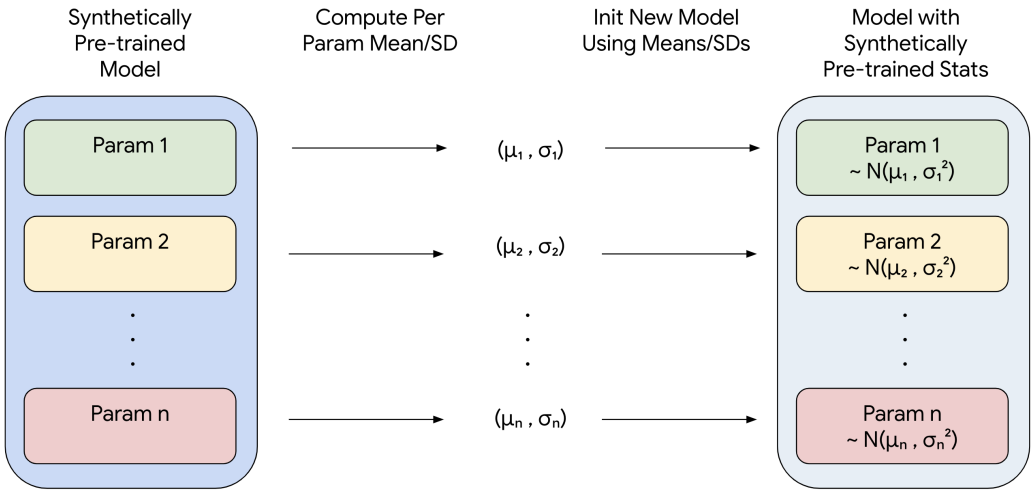

Figure 4: Illustration of Section 5.1 `Per_Param_Mean_SD` init scheme. We use this init scheme with `LIME`, `Set`, or `Identity` pre-trained models before fine-tuning on the downstream tasks. As an example, if we were to use this init scheme with an `Identity` pre-trained model, we would initialize the encoder layer 2 query matrix weights by sampling from a Gaussian with the the mean and SD of the encoder layer 2 query matrix from an `Identity` pre-trained model, and we would initialize every other parameter in the same manner. The `Per_Param_Mean_SD` init scheme recovered more than 70% of the benefit of normally fine-tuning a synthetic pre-trained model (or an average of 39% of natural pre-training benefits).

5.1 How much of synthetic pre-training benefit can be attributed to better statistics for initialization?

We compare normal fine-tuning, which we call `Full_Init`, with fine-tuning a model that is initialized with the mean and standard deviation (SD) of each synthetic pre-trained model's parameter. We call this initialization scheme `Per_Param_Mean_SD` and illustrate it in Figure 4. Specifically, for each parameter p (e.g. an attention query matrix) in the synthetic pre-trained model, we compute the the mean μ_p and the standard deviation σ_p of all entries in p. We then randomly initialize the model with the same μ_p and σ_p for each parameter p. For example, if a pre-trained model had a 2×2 parameter matrix $p = [[-1, 2], [2, 5]]$, then $\mu_p = 2$ and $\sigma_p = 2.4$, and we would initialize p by sampling the each entry of this 2×2 matrix from a normal distribution $\mathcal{N}(2, 2.4^2)$. We list all all the parameters we initialize for T5-Small in this manner in Appendix K.

As shown in Table 2, using the `Per_Param_Mean_SD` initialization scheme achieves a large proportion of the benefit from normally fine-tuning a synthetic pre-trained model. For example, for `LIME` `Identity` and `Set`, the `Per_Param_Mean_SD` initialization recovered 78%, 85%, and 97%[2] of MTOP `Full_Init` performance (or 65%, 66%, 69% of the benefits by natural pre-training). The remaining gap between `Per_Param_Mean_SD` and `Full_Init` supports the possibility that other factors besides better initialization statistics may cause the benefits of synthetic pre-training. We note that it does not preclude the possibility that better initialization statistics is the sole cause of benefits: there may exist other initialization schemes such that the statistics of the pre-trained model are a subset of the statistics from the initialization scheme.

5.2 Can simpler initialization schemes still produce benefits?

Next we explore if simpler initialization schemes that use a smaller number of statistics than `Per_Param_Mean_SD` can still provide benefits. We present three initialization schemes: `Per_Param_Scale`, `Whole_Model_Scale`, and `Subset_Per_Param_Scale`, with results shown in Table 2. We also tried two other initialization schemes, `Across_Layers_Scale` and `Per_Layer_Scale`, discussed in Appendix H.

[2]These percentages are computed by $\frac{\text{performance(Per_Param_Mean_SD)} - \text{performance(Random_Init)}}{\text{performance(Full_Init)} - \text{performance(Random_Init)}}$

Table 2: Evaluating different initialization schemes with statistics of LIME, Set, and Identity pre-trained models. Results for Per_Param_Mean_SD for the four other tasks as well as five seeds for CNNDM-10K, MTOP, and SQuAD are in Appendix Table 7. Results for two other initialization ablations and three other Subset_Per_Param_Scale subset choices we tried are in Appendix Table 5.

	CNNDM-10K			MTOP		
	LIME	Set	Identity	LIME	Set	Identity
Section 5.1						
Full_Init	33.2	32.8	30.2	73.7/94.0	71.7/93.7	68.6/93.0
Per_Param_Mean_SD	29.3	29.8	28.8	66.1/92.3	66.6/92.3	67.8/92.6
Section 5.2						
Per_Param_Scale	29.0	29.5	29.0	67.3/92.5	66.5/92.4	66.3/92.3
Whole_Model_Scale	15.5	26.1	20.5	34.3/81.4	38.7/83.1	59.8/91.0
Pre-attn_LN_Per_Param_Scale	27.8	25.1	27.3	59.9/90.3	58.9/89.0	46.7/86.3

Baseline	CNNDM-10K	MTOP
Random_Init	18.9	38.6/83.0
Pre-trained_T5	35.8	81.0/95.2

Per Param Scale We use the term "scale statistic" to denote the mean of layer norm parameters or the SD of non layer norm parameters. Unlike the Per_Param_Mean_SD init scheme in the previous subsection, where we initialize each parameter with two statistics (mean and SD), Per_Param_Scale initializes each parameter with only the scale statistic: initialize layer norm parameters by setting their value to the mean of the corresponding pre-trained parameter weights, and initialize non layer norm parameters by sampling from a Gaussian with mean 0 and the SD of the corresponding pre-trained parameter weights.

Whole Model Scale We initialize the whole model with only two statistics: the SD over all non layer norm weights σ_m and the mean over all layer norm weights μ_m. Initialize each non layer norm weight by sampling from $\mathcal{N}(0, \sigma_m^2)$ and Initialize each layer norm weight as the value μ_m.

Per Param Scale: Subset The motivation for this init scheme is to find a smaller set of parameters whose initialization can provide significant benefit. We initialize a subset of the parameters (e.g., attention parameters) with the Per_Param_Scale initialization and initialize the rest of the parameters with the default Random_Init. With this initialization scheme, we tried four different subsets: attention parameters, MLP parameters, pre-attention layer norms, and pre-MLP layer norms.

Results Per_Param_Scale provided almost identical benefits as the Per_Param_Mean_SD initialization, suggesting that just the scale statistics, rather than both the mean and SD, are sufficient to provide benefits of initialization.

Whole_Model_Scale initialization was much worse than Per_Param_Mean_SD, yielding almost no improvements. A finer scale initialization rather than simply increasing the scale of weights across the whole model as Whole_Model_Scale does may be necessary for benefits.

For the Subset_Per_Param_Scale init scheme, pre-attention layer norms was the only subset we tried that provided significant initialization benefits, covering 46% and 39% of the gap between From Scratch and Pre-trained_T5 for CNNDM-10K and MTOP. See results in Table 2 (labeled Pre-attn_LN_Per_Param_Scale) and results for all four subsets tried in Appendix Table 5.

5.3 A Better Pre-attention Layer Norm Initialization

Synthetic pre-training results in lower pre-attention layer norm means. Motivated by benefits from the Subset_Per_Param_Scale initialization scheme—initializing pre-attention layer norms using their synthetic pre-trained means and all other parameters with the default initialization—we wanted to see if the pre-attention layer norm mean values in synthetic pre-trained models have a

Table 3: Results from initializing pre-attention layer norm parameters to a particular value. They are initialized to the value of 1.0 in the `Random_Init` baseline which is the default T5 initialization.

Pre-attn LN Init Value	CNNDM-10K	MTOP	WebQSP	SQuAD	Code Trans.	Retrosyn.
0.05	28.2	25.7/73.3	30.5/71.2	22.8/31.2	57.9	39.3
0.1	28.3	32.5/77.5	28.9/70.6	23.5/31.7	57.6	39.9
0.2	28.2	56.4/89.3	30.6/72.3	28.2/37.3	57.0	39.4
0.4	24.4	56.9/89.9	30.0/72.8	36.3/46.9	57.3	39.6
0.8	18.6	50.7/88.5	28.0/71.6	34.2/44.6	57.2	39.9

Baseline	CNNDM-10K	MTOP	WebQSP	SQuAD	Code Trans.	Retrosyn.
`Random_Init`	18.9 (0.5)	38.6/83.0 (2.1/1.2)	25.8/71.5 (0.7/0.5)	16.0/24.2 (2.1/2.0)	57.6 (0.3)	39.0 (0.5)
`Pre-trained_T5`	35.8 (0.1)	81.0/95.2 (0.3/2.2)	82.3/91.7 (0.8/0.4)	77.4/86.1 (0.2/0.2)	61.2 (0.7)	43.5 (0.3)

pattern. In Appendix Figure 6, we plotted pre-attention layer norm parameter means of each layer for `LIME`, `Set`, and `Identity` pre-trained models. We noticed that the pre-attention layer norm means from the synthetic pre-trained models are much smaller values (below 0.4 for most layers) compared to the default initialization of 1.0.

Lower value pre-attention layer norm initialization provides benefits. Inspired by the above observation, we tried an initialization scheme where all the pre-attention layer norm parameters are initialized to a specific value, and all other parameters are initialized with the default initialization. We tried five different values ranging from 0.05 to 0.8. Results in Table 3 show that initializing the pre-attention layer norm parameters to a lower value than the default initialization of 1.0 provided significant benefits: initializing them to 0.2 improved CNNDM-10K from 18.9 to 28.2 ROUGE1, MTOP from 38.6% to 56.4% EM, and SQuAD from 16.0 to 22.8 EM.

6 Related Work

Synthetic Pre-training Past work has shown that synthetic pre-training can benefit a wide range of downstream tasks. For example, it has been shown that pre-training on artificial languages that intuitively mimic properties of natural language can benefit language modeling, dependency parsing, and GLUE (Ri & Tsuruoka, 2022; Papadimitriou & Jurafsky, 2020; Chiang & Lee, 2021). Pre-training on LIME, a mixture of three synthetic tasks motivated from reasoning primitives, benefits mathematical reasoning benchmarks (Wu et al., 2021). For summarization, Krishna et al. (2021) designed a suite of synthetic summarization tasks on a corpus of random tokens, and they show such pre-training can almost match up with natural pre-training. Synthetic pre-training also has been applied outside of text-based domains: Anderson & Farrell (2022) shows pre-training on synthetically generated fractal images nearly matches the benefits of pre-training with ImageNet.

Cross-domain Transfer Learning Synthetic pre-training benefiting downstream tasks across a wide range of domains is an instance of cross-domain transfer learning, which past work has demonstrated the efficacy of. Pre-training language models with many languages benefits downstream tasks in a new unseen language (Liu et al., 2020). More surprisingly, pre-training on music, code, and amino acid sequences benefits natural language downstream tasks (Papadimitriou & Jurafsky, 2020; Chiang & Lee, 2020). It was also shown recently that pre-training on natural language benefits offline RL (Reid et al., 2022).

Understanding Pre-training Past work has explored what properties of the pre-training data are important for benefits. Ri & Tsuruoka (2022) and Papadimitriou & Jurafsky (2020) use synthetic pre-training to identify general properties of pre-training data that correlate with improved downstream language task performance by leveraging how synthetic data is easy to modify in a controlled manner. Papadimitriou & Jurafsky (2020) found that changing the distribution of tokens in an artificial language can affect downstream performance, and Ri & Tsuruoka (2022) found that for an artificial brackets language using a nesting structure as well as different tokens for bracket pairs results in better performance than using a flat structure and the same token for bracket pairs. K et al. (2020), Conneau et al. (2020), and Dufter & Schütze (2020) explore data properties related to cross-language transfer learning. Sinha et al. (2021) shows that word order within sentences is not essential for natural langauge pre-training benefits.

Other past work has used probing methods to explore what potentially transferable knowledge is gained from pre-training. Ri & Tsuruoka (2022) explored how much knowledge of "position-aware context dependence" was gained from artificial language and natural language pre-training. Chi et al. (2020) and Papadimitriou et al. (2021) explored potentially transferable knowledge gained from multi-lingual language pre-training.

Initialization Our work tried to understand pre-training through the lens of initialization statistics that benefit optimization. There has been past work on improving transformer initialization to achieve more stable training and benefits with deeper models (Zhang et al., 2019; Huang et al., 2020; Zhang et al., 2020; Wang et al., 2022). These papers do not explore benefits from initializing with pre-trained model statistics.

7 Discussion

Limitations of current synthetic tasks In this work, we show that synthetic pre-training can provide significant benefits across many different tasks. However, synthetic pre-training still lags behind natural pre-training by an average of 33% for the six downstream tasks we fine-tuned on. This is expected for natural language downstream tasks, but it is important to note that even on non-natural language tasks, natural language pre-training still outperforms synthetic pre-training. This result suggests that natural language pre-training produces effects that are missing from the evaluated synthetic pre-training. Exploring the differences between different pre-training may be an interesting direction for future work.

We emphasize that this paper does not support any claims regarding the practical efficacy of synthetic pre-training. The metric of "percentage of natural pre-training gap closed" measures relative performance along a meaningless linear scale, and for downstream tasks retrosynthesis and code translation, the magnitude of the improvement over from scratch for natural pre-training is low to begin with. This paper's goal is not to communicate that synthetic pre-training is practically effective, but rather to communicate that synthetic pre-training produces nontrivial differences in model behavior compared to a randomly initialized model, and also that there is nontrivial variation in model behavior from differing synthetic pre-training and initialization schemes involving pre-trained weight statistics.

Synthetic pre-training requires deeper understanding Our results show that initialization using the statistics from synthetic pre-trained models can recover more than 70% of the benefit from synthetic pre-training. The remaining gap leaves open the possibility that the benefits of synthetic pre-training are due to other factors besides better statistics for initialization. Due to their simplicity, `Set` and `Identity` may be useful for future work towards deeper understanding of synthetic pre-training.

In the long term, as we learn more about synthetic pre-training and pre-training in general, we believe that for some downstream tasks it may be possible to develop synthetic pre-training that outperforms natural pre-training: the complexity of existing natural data is fixed, while in some sense the complexity of fully synthetically generated data is infinite.

Privacy and Ethics This work focuses on the understanding of synthetic pre-training. But synthetic pre-training can also have practical implications for privacy and ethics. Standard pre-training data, even when its public, can contain private user information. Large pre-trained models are capable of memorizing training examples, which makes them vulnerable to privacy attacks (Carlini et al., 2020). Synthetic data on the other hand is divorced from the real world, which can help mitigate privacy issues. Large pre-trained corpora can contain harmful text (e.g., hate speech). With synthetic pre-training, since we have complete control over what goes into the data, we can potentially mitigate some of the harmful behavior of existing models. Unfortunately, synthetic pre-training is currently not as performant as natural pre-training, so real-world data would still have to be used to close the performance gap.

Acknowledgements

We thank Google TPU Research Cloud for the experimental support. We thank Christian Szegedy, Eric Zelikman, Tri Dao, Dan Fu, Sidd Karamcheti and Rishi Bommasani, for their useful feedbacks on the draft.

References

Connor Anderson and Ryan Farrell. Improving fractal pre-training. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pp. 2412–2421. IEEE, 2022. doi: 10.1109/WACV51458.2022.00247. URL https://doi.org/10.1109/WACV51458.2022.00247.

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

Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom B. Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. *CoRR*, abs/2012.07805, 2020. URL https://arxiv.org/abs/2012.07805.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

Ethan A. Chi, John Hewitt, and Christopher D. Manning. Finding universal grammatical relations in multilingual BERT. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 5564–5577. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.493. URL https://doi.org/10.18653/v1/2020.acl-main.493.

David Cheng-Han Chiang and Hung-yi Lee. Pre-training a language model without human language. *CoRR*, abs/2012.11995, 2020. URL https://arxiv.org/abs/2012.11995.

David Cheng-Han Chiang and Hung-yi Lee. On the transferability of pre-trained language models: A study from artificial datasets. *CoRR*, abs/2109.03537, 2021. URL https://arxiv.org/abs/2109.03537.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 8440–8451. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.747. URL https://doi.org/10.18653/v1/2020.acl-main.747.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *ACL*, 2019.

Philipp Dufter and Hinrich Schütze. Identifying elements essential for bert's multilinguality. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 4423–4437. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.358. URL https://doi.org/10.18653/v1/2020.emnlp-main.358.

Mandy Guo, Zihang Dai, Denny Vrandečić, and Rami Al-Rfou. Wiki-40B: Multilingual language model dataset. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 2440–2452, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.297.

Xiao Shi Huang, Felipe Pérez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4475–4483. PMLR, 2020. URL http://proceedings.mlr.press/v119/huang20f.html.

Karthikeyan K, Zihan Wang, Stephen Mayhew, and Dan Roth. Cross-lingual ability of multilingual BERT: an empirical study. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HJeT3yrtDr.

Kundan Krishna, Jeffrey P. Bigham, and Zachary C. Lipton. Does pretraining for summarization require knowledge transfer? In *EMNLP (Findings)*, pp. 3178–3189, 2021. URL https://aclanthology.org/2021.findings-emnlp.273.

Haoran Li, Abhinav Arora, Shuohui Chen, Anchit Gupta, Sonal Gupta, and Yashar Mehdad. MTOP: A comprehensive multilingual task-oriented semantic parsing benchmark. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 2950–2962, Online, April 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.eacl-main.257. URL https://aclanthology.org/2021.eacl-main.257.

Bowen Liu, Bharath Ramsundar, Prasad Kawthekar, Jade Shi, Joseph Gomes, Quang Luu Nguyen, Stephen Ho, Jack Sloane, Paul Wender, and Vijay Pande. Retrosynthetic reaction prediction using neural sequence-to-sequence models. *ACS Central Science*, 3(10):1103–1113, 2017. doi: 10.1021/acscentsci.7b00303. URL https://doi.org/10.1021/acscentsci.7b00303. PMID: 29104927.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Trans. Assoc. Comput. Linguistics*, 8:726–742, 2020. URL https://transacl.org/ojs/index.php/tacl/article/view/2107.

Jieyu Lu and Yingkai Zhang. Unified deep learning model for multitask reaction predictions with explanation. *Journal of Chemical Information and Modeling*, 62(6):1376–1387, 2022. doi: 10.1021/acs.jcim.1c01467. URL https://doi.org/10.1021/acs.jcim.1c01467. PMID: 35266390.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. Codexglue: A machine learning benchmark dataset for code understanding and generation. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c16a5320fa475530d9583c34fd356ef5-Abstract-round1.html.

Isabel Papadimitriou and Dan Jurafsky. Learning music helps you read: Using transfer to study linguistic structure in language models. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pp. 6829–6839. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.emnlp-main.554. URL https://doi.org/10.18653/v1/2020.emnlp-main.554.

Isabel Papadimitriou, Ethan A. Chi, Richard Futrell, and Kyle Mahowald. Deep subjecthood: Higher-order grammatical features in multilingual BERT. In Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty (eds.), *Proceedings of the 16th Conference of the European Chapter of the Association for*

Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021, pp. 2522–2532. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.eacl-main.215. URL https://doi.org/10.18653/v1/2021.eacl-main.215.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL http://arxiv.org/abs/1606.05250.

Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *CoRR*, abs/2201.12122, 2022. URL https://arxiv.org/abs/2201.12122.

Ryokan Ri and Yoshimasa Tsuruoka. Pretraining with artificial language: Studying transferable knowledge in language models. *CoRR*, abs/2203.10326, 2022. doi: 10.48550/arXiv.2203.10326. URL https://doi.org/10.48550/arXiv.2203.10326.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1099. URL https://aclanthology.org/P17-1099.

Koustuv Sinha, Robin Jia, Dieuwke Hupkes, Joelle Pineau, Adina Williams, and Douwe Kiela. Masked language modeling and the distributional hypothesis: Order word matters pre-training for little. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 2888–2913. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.230. URL https://doi.org/10.18653/v1/2021.emnlp-main.230.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1, 000 layers. *CoRR*, abs/2203.00555, 2022. doi: 10.48550/arXiv.2203.00555. URL https://doi.org/10.48550/arXiv.2203.00555.

Yuhuai Wu, Markus N Rabe, Wenda Li, Jimmy Ba, Roger B Grosse, and Christian Szegedy. Lime: Learning inductive bias for primitives of mathematical reasoning. In *International Conference on Machine Learning*, pp. 11251–11262. PMLR, 2021.

Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir R. Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. *CoRR*, abs/2201.05966, 2022. URL https://arxiv.org/abs/2201.05966.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 201–206, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-2033. URL https://aclanthology.org/P16-2033.

Biao Zhang, Ivan Titov, and Rico Sennrich. Improving deep transformer with depth-scaled initialization and merged attention. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 898–909. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1083. URL https://doi.org/10.18653/v1/D19-1083.

Biao Zhang, Philip Williams, Ivan Titov, and Rico Sennrich. Improving massively multilingual neural machine translation and zero-shot translation. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pp. 1628–1639. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.acl-main.148. URL https://doi.org/10.18653/v1/2020.acl-main.148.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Sections 2,3.1,4.1,G,D
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Appendix Section I
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We release the source code at https://github.com/felixzli/synthetic_pretraining.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Synthetic Pre-training Details

We do synthetic pre-training with the same hyperparameters that the off-the-shelf pre-trained T5-small was trained with: AdaFactor optimizer, batch size 128, sequence length 512, and inverse square root learning rate $1/\sqrt{max(n, 10000)}$ where n is the current step. We evaluate token validation accuracy and save a checkpoint every 5000 steps. For synthetic tasks besides the artificial language, we fine-tuned with the first checkpoint that the model reaches above 99% token validation accuracy. For LIME, Nonsense Summary, Set, and Identity this was 30K steps, 5K steps, 10K steps, and 5K steps respectively. For the artificial language pre-training, after training for 300K steps the model's token validation accuracy had plateaued without converging to above 99% token validation accuracy, so we chose to fine-tune with the max accuracy checkpoint, which was at step 165K with accuracy of 77.7%. For reference, off-the-shelf pre-trained T5-small was trained for 524,288 steps (Raffel et al., 2020).

B Fine-tuning Details

We fine-tuned with the same hyperparameters (Raffel et al., 2020) fine-tuned with, with the exception of learning rate, for which we do a sweep over 1e-2, 3e-3, 1e-3. We use the AdaFactor optimizer and batch size 128 for all except WebQSP and Code Translation, for which we use batch size 32 due to their long sequence length. We list the learning rate and sequence length for each task in Appendix D.3. We use the default T5 tokenization (Raffel et al., 2020). We fine-tuned for a max of 150K steps for code translation, 400K steps for retrosynthesis, 250K steps for SQuAD, 50K steps for WebQSP, 100K steps for MTOP, and 100K steps for CNNDM-10K. We evaluate and save checkpoints every 10K steps for all tasks besides WebQSP, for which we evaluate and save checkpoints every 5K steps. For all tasks besides CNNDM-10K and SQuAD, we report the test set metric achieved with the checkpoint corresponding to the max token validation accuracy checkpoint. For CNNDM-10K, we reported the test set metric achieved at step 50K, because we observed the test set metric achieved with the max token validation accuracy checkpoint was often significantly worse than later step checkpoints, and the test set metric always plateaued before and up to about 50K steps. For SQuAD, we report the best validation set metric, as was done in (Raffel et al., 2020), because evaluating on the test set requires running inference on a benchmark server.

C Off-the-shelf Pre-trained T5v1.1-Small for CNNDM-10K

The differences between Pre-trained T5v1.1 and Pre-trained T5 are:

1. T5v1.1 Uses the GEGLU activation in feed-forward hidden layers rather than ReLU.
2. Dropout was turned off in pre-training for T5v1.1.
3. No parameter sharing between the embedding and classifier layer for T5v1.1.
4. **T5v1.1 was pre-trained on C4 only, without mixing in the downstream tasks.**

The last difference is why we fine-tune CNNDM-10K with Pre-trained T5v1.1 instead of Pre-trained T5. Using Pre-trained T5 would be an unfair comparison because it has already trained been on all 290K CNNDM task training data.

D Downstream Tasks

D.1 Task Descriptions

CNNDM-10K is 10K training examples from the CNNDM benchmark, which consists of news articles from CNN and Daily Mail and summaries of each article (See et al., 2017). We evaluated on CNNDM-10K instead of the full CNNDM because this was the dataset that (Krishna et al., 2021) evaluated their nonsense summarization synthetic task on, which was one of the synthetic tasks we evaluate in Section 3.

MTOP (Li et al., 2021) and **WebQSP** (Yih et al., 2016) are two benchmarks for semantic parsing, the task of converting a natural language query to a logical form. For these tasks, we use the data from (Xie et al., 2022) that is already processed to be in a Seq2Seq format compatible with T5. MTOP has 17K training examples and WebQSP has 2.7K training examples.

SQuAD 1.1 is a reading comprehension dataset consisting of "questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text or span from the corresponding reading passage" (Rajpurkar et al., 2016). There are 87K training examples.

Java to C# code translation is one task within Microsoft's CodeXGLUE: General Language Understanding Evaluation benchmark for Code (Lu et al., 2021). The task consists of 10K training examples that were collected from the code of several open-source projects that were originally developed in Java and then ported to C#.

USPTO-50K is a dataset of 50K chemical reactions that is a commonly used to benchmark deep learning methods for the chemistry task of single-step retrosynthesis, which is predicting possible reactants when given a product as input. (Liu et al., 2017). For this task, we use the data from (Lu & Zhang, 2022) that is already processed to be in a Seq2Seq format compatible with T5. There are 40K training examples.

D.2 Task Examples

```
Source:
summarize: marouane fellaini and adnan januzaj continue to show the world they are not just teammates but
      also best mates . the manchester united and belgium duo both posted pictures of themselves out at a
      restaurant on monday night ahead of their game against newcastle on wednesday . januzaj poses in the
      middle of fellaini and a friend looking like somebody who failed to receive the memo about it being a
       jackson 5 themed night . premier league duo adnan januzaj and marouane fellaini pose with a friend
      on the dance floor . manchester united and belgium duo fellaini and januzaj are good friends both on
      and off the pitch . manchester united ace fellaini runs over to the bench to celebrate his goal
      against qpr with friend januzaj . the disco effect in the background adds to the theory, but januzaj
      doesn't seem to mind as they later pose on the dance floor with other friends. united haven't had too
       many reasons to have a song and dance this season so it seems they may be hitting the discotheques
      as another form of release . however, victory against newcastle on wednesday would leave manager
      louis van gaal at least tapping his toes as they continue to fight for a champions league spot this
      season. januzaj and robin van persie join fellaini in celebrating in front of the manchester united
      fans at west brom . januzaj receives some words of wisdom from manchester united's dutch manager
      louis van gaal . januzaj and fellaini are joined by some friends as they take to the dance floor
      ahead of the newcastle game .

Target:
the belgian duo took to the dance floor on monday night with some friends . manchester united face
      newcastle in the premier league on wednesday . red devils will be looking for just their second
      league away win in seven . louis van gaal's side currently sit two points clear of liverpool in
      fourth .
```

```
Source:
Has Angelika Kratzer video messaged me ? ; structured knowledge: IN:GET: MESSAGE, WEATHER, ALARM, INFO_
      RECIPES, STORIES_NEWS, REMINDER, RECIPES, EVENT, CALL_TIME, LIFE_EVENT, INFO_CONTACT, CONTACT, TIMER,
       REMINDER_DATE_TIME, AGE, SUNRISE, EMPLOYER, EDUCATION_TIME, JOB, AVAILABILITY, CATEGORY_EVENT, CALL,
       EMPLOYMENT_TIME, CALL_CONTACT, LOCATION, TRACK_INFO_MUSIC, SUNSET, MUTUAL_FRIENDS, UNDERGRAD,
      REMINDER_LOCATION, ATTENDEE_EVENT, MESSAGE_CONTACT, REMINDER_AMOUNT, DATE_TIME_EVENT, DETAILS_NEWS,
      EDUCATION_DEGREE, MAJOR, CONTACT_METHOD, LIFE_EVENT_TIME, LYRICS_MUSIC, AIRQUALITY, LANGUAGE, GENDER,
       GROUP | IN:SEND: MESSAGE | IN:SET: UNAVAILABLE, RSVP_YES, AVAILABLE, DEFAULT_PROVIDER_MUSIC, RSVP_
      INTERESTED, DEFAULT_PROVIDER_CALLING, RSVP_NO | IN:DELETE: REMINDER, ALARM, TIMER, PLAYLIST_MUSIC |
      IN:CREATE: ALARM, REMINDER, CALL, PLAYLIST_MUSIC, TIMER | IN:QUESTION: NEWS, MUSIC | IN:PLAY: MUSIC,
      MEDIA | IN:END: CALL | IN:IGNORE: CALL | IN:UPDATE: CALL, REMINDER_DATE_TIME, REMINDER_TODO, TIMER,
      METHOD_CALL, ALARM, REMINDER_LOCATION, REMINDER | IN:PAUSE: MUSIC, TIMER | IN:ANSWER: CALL | IN:
      SNOOZE: ALARM | IN:IS: TRUE_RECIPES | IN:REMOVE: FROM_PLAYLIST_MUSIC | IN:ADD: TIME_TIMER, TO_
      PLAYLIST_MUSIC | IN:SHARE: EVENT | IN:PREFER:  | IN:START: SHUFFLE_MUSIC | IN:SILENCE: ALARM | IN:
      SWITCH: CALL | IN:SUBTRACT: TIME_TIMER | IN:PREVIOUS: TRACK_MUSIC | IN:HOLD: CALL | IN:SKIP: TRACK_
      MUSIC | IN:LIKE: MUSIC | IN:RESTART: TIMER | IN:RESUME: TIMER, CALL, MUSIC | IN:MERGE: CALL | IN:
      REPLAY: MUSIC | IN:LOOP: MUSIC | IN:STOP: MUSIC, SHUFFLE_MUSIC | IN:UNLOOP: MUSIC | IN:CANCEL:
      MESSAGE, CALL | IN:REWIND: MUSIC | IN:REPEAT: ALL_MUSIC, ALL_OFF_MUSIC | IN:FAST: FORWARD_MUSIC | IN:
      DISLIKE: MUSIC | IN:DISPREFER:  | IN:HELP: REMINDER | IN:FOLLOW: MUSIC

Target:
[IN:GET_MESSAGE [SL:CONTACT Angelika Kratzer ] [SL:TYPE_CONTENT video ] [SL:RECIPIENT me ] ]
```

```
Source:
where is isthmus of panama located? ; structured knowledge: Isthmus of Panama: m.04zwft | m.06n3y location.
      location.contains m.06w99sr | m.06n3y location.location.contains m.0cn1d6 | m.06n3y location.location.
      contains m.0c_ys | m.06n3y location.location.contains m.03718t | m.06n3y location.location.contains m
      .0w_hmnw | m.06n3y location.location.contains m.0c4rq4 | m.06n3y location.location.contains m.02t1x0
      | m.06n3y location.location.contains m.05w5vr | m.04zwft common.topic.notable_types m.01nt | m.06n3y
      location.location.contains m.09vgjp | m.06n3y location.location.contains m.0b2ft5 | m.06n3y location.
      location.contains m.07twz | m.06n3y location.location.contains m.0cqy3y | m.04zwft location.location.
      containedby m.06n3y | m.06n3y location.location.contains m.02x1y_b | m.06n3y location.location.
      contains m.03t3qb | m.06n3y location.location.contains m.06w57th | m.06n3y location.location.contains
       m.06w9q8y | m.06n3y location.location.contains m.0cpbxws | m.06n3y location.location.contains m.07t1
      sw | m.06n3y location.location.contains m.01nr2h | m.06n3y location.location.contains m.06gldq | m.06
      n3y location.location.contains m.034m8 | m.06n3y location.location.contains m.0dkz7x | m.06n3y
      location.location.contains m.0c7r4h | m.06n3y location.location.contains m.025vjdw | m.06n3y location.
      location.contains m.02qbjjz | m.06n3y location.location.contains m.0wq95z_ | m.06n3y location.
      location.contains m.0sd7 | m.06n3y location.location.contains m.06bf8s | m.06n3y location.location.
      contains m.027d7t4 | m.06n3y location.location.contains m.0cn1vs | m.06n3y location.location.contains
       m.076ycbb | m.06n3y location.location.contains m.061b7x | m.06n3y location.location.contains m.0p2n
      | m.06n3y location.location.contains m.0w_j2g5 | m.06n3y location.location.contains m.015fr | m.06n3y
       location.location.contains m.02xsr6 | m.06n3y location.location.contains m.02z722h | m.06n3y
      location.location.contains m.06w3qq8 | m.06n3y location.location.contains m.0cn1k1 | m.06n3y location.
      location.contains m.02vrgz0 | m.06n3y location.location.contains m.0g61h6 | m.06n3y location.location.
      contains m.0w_j7px | m.06n3y location.location.contains m.05fzj6 | m.06n3y location.location.contains
       m.026h5zg | m.06n3y location.location.contains m.0jgd | m.06n3y location.location.contains m.03qjmgc
       | m.06n3y location.location.contains m.01ls2 | m.06n3y location.location.containedby m.07c5l

Target:
(JOIN (R location.location.containedby) m.04zwft)
```

```
Source:
question: What does increased oxygen concentrations in the patient's lungs displace? context: Hyperbaric (
      high-pressure) medicine uses special oxygen chambers to increase the partial pressure of O 2 around
      the patient and, when needed, the medical staff. Carbon monoxide poisoning, gas gangrene, and
      decompression sickness (the 'bends') are sometimes treated using these devices. Increased O 2
      concentration in the lungs helps to displace carbon monoxide from the heme group of hemoglobin.
      Oxygen gas is poisonous to the anaerobic bacteria that cause gas gangrene, so increasing its partial
      pressure helps kill them. Decompression sickness occurs in divers who decompress too quickly after a
      dive, resulting in bubbles of inert gas, mostly nitrogen and helium, forming in their blood.
      Increasing the pressure of O 2 as soon as possible is part of the treatment.

Target:
carbon monoxide
```

```
Source:
public void delete(int key) {int i = binarySearch(mKeys, 0, mSize, key);if (i >= 0) {if (mValues[i] !=
      DELETED) {mValues[i] = DELETED;mGarbage = true;}}}

Target:
public virtual void delete(int key){int i = binarySearch(mKeys, 0, mSize, key);if (i >= 0){if (mValues[i]
      != DELETED){mValues[i] = DELETED;mGarbage = true;}}}
```

```
Source:
COc1ccc(CN(C(=O)OCc2ccccc2)[C@@H]2C(=O)N(Cc3ccc(OC)cc3OC)[C@@H]2CC=C(Br)Br)cc1

Target:
BrC(Br)(Br)Br.COc1ccc(CN(C(=O)OCc2ccccc2)[C@@H]2C(=O)N(Cc3ccc(OC)cc3OC)[C@@H]2CC=O)cc1
```

D.3 Downstream Task Specific Hyperparameters

More general fine-tuning hyperparameters and details are in Appendix B. For each task, we chose the learning rate by trying 1e-2, 3e-3, 1e-3 for Random Init finetuning, and picking the best performing value. .

CNNDM-10K We use the same input length, taget length, and max decoding length parameters as the Nonsense Summary synthetic task paper (Krishna et al., 2021).

- learning rate: 0.001
- input length: 512
- target length: 256
- max decode length: 148

MTOP

- learning rate: 0.001
- input length: 1024
- target length: 128

WebQSP

- learning rate: 0.001
- input length: 2048
- target length: 256

SQuAD

- learning rate: 0.001
- input length: 512
- target length: 128

Code Translation

- learning rate: 0.003
- input length: 1024
- target length: 1024

Retrosynthesis

- learning rate: 0.003
- input length: 256
- target length: 256

E Three Previously Proposed Synthetic Tasks from Section 3

E.1 LIME

The input token length ranges from 68 to 214 and the output token length ranges from 7 to 74. For the following LIME examples, the "rule string" is $A * A + B = C$, the "case dictionary" that represents substitutions is $\{A : a, B : b, C : d + e\}$, and the "result string" is $a * a + b = d + e$. In real generated data, the "Rule Symbols" and "Math Symbols" are randomly sampled from a vocabulary of 32K tokens for each data point.

```
Source:
<RuleSymbols> A B C <MathSymbols> * + = a b d e <s> {A : a, B : b, C : d + e} <s> a * a + b = d + e

Target:
A * A + B = C
```

```
Source:
<RuleSymbols> A B C <MathSymbols> * + = a b d e <s> A * A + B = C <s> a * a + b = d + e

Target:
{A : a, B : b, C : d + e}
```

```
Source:
<RuleSymbols> A B C <MathSymbols> * + = a b d e <s> A * A + B = C <s> {A : a, B : b, C : d + e}

Target:
a * a + b = d + e
```

E.2 Dyck Artificial Language

The input token length ranges from 3 to 512 and the output token length ranges from 3 to 114. For clarity, the following example will involve only 3 head and tail tokens represented by the symbols {, }, [,], (,). In real generated data, there will be 16K possible head-tail token pairs.

An example sentence:

```
[ ( { [ ] } [ ] ) ] { [ ] }
```

This sentence converted to a Seq2Seq task with the T5 pre-training objective:

```
Source:
[ <s0> <s1> [ ] } <s2> ] ) <s3> { [ ] <s4>

Target:
( <s0> { <s1> [ <s2> ] <s3> }
```

E.3 Nonsense Summary

The input token length ranges from 175 to 709 and the output token length ranges from 12 to 825. We provide an example of a nonsense summary data point in Appendix Figure 5, which is directly taken from the task's original paper (Krishna et al., 2021).

F Dataset Size Ablation Full Results

The table showing the full training dataset size ablation results is in Appendix Table 4.

G Simpler Synthetic Tasks from Section 4

All data is generated by sampling over 32K tokens. But for clarity, all examples are written with individual alphabetic characters representing tokens. All simpler synthetic tasks have input between length 10 and length 220.

G.1 Set

Definition Input is a token sequence. Output is the input token sequence with no duplicates (in original order).

```
Source:
a b a a b b c d

Target:
a b c d
```

Input Generation To generate one data point, we do not just uniformly sample tokens. Because over a vocabulary of 32K tokens, that simple sampling would result in few sequences with duplicate tokens where the task input and output are different. Instead, we uniformly sample the length l of the input from min length to max length, then sample t tokens that will be in the input where $t < l$, then sample the amount of each token by randomly partitioning l to t parts $\{p_1, p_2...p_t\}$, then finally generate a random input sequence based on the t sampled tokens and corresponding amounts.

G.2 Delete

Definition Input is token sequence 1 and token sequence 2. Output is sequence 1 with the first occurrence of sequence 2 deleted.

```
Source:
c b a a b b a d <s> b a

Target:
b a b b a d
```

Input Generation With 0.75 probability, generate sequence 1 and sequence 2 such that sequence 2 appears in sequence 1 at least once via the following process: sample sequence 2, sample sequence 1 length, sample location that sequence 2 appears in sequence 1, sample remaining tokens in sequence 1. With 0.25 probability, generate sequence 1 and sequence 2 such that sequence 2 does not appear in sequence 1 via the following process: sample sequence 1, sample sequence 2 and keep re-sampling sequence 2 until it does not appear in sequence 1.

G.3 Sort

Definition Input is token sequence 1 and token sequence 2. Output is input sequence 1 sorted in according to order specified by token sequence 2.

```
Source:
c b a a b c b a d <s> b a c d

Target:
b b b a a a c c d
```

Input Generation To generate token sequence 1, uniformly sample the length l of the input from min length to max length, then sample t tokens that will be in the input where $t < l$, then sample the amount of each token by randomly partitioning l to t parts $\{p_1, p_2...p_t\}$, then finally generate a random input sequence based on the t sampled tokens and corresponding amounts. To generate sequence 2, shuffle the set of tokens in sequence 1.

G.4 Identity

Definition Input is a token sequence. Output is the same as the input token sequence.

```
Source:
c b a a b

Target:
c b a a b
```

Input Generation Sample input sequence length l uniformly between min and max possible length, then sample l tokens.

G.5 Union

Definition Input is token sequence 1 and token sequence 2. Output is token sequence that is the union of the set of tokens in sequence 1 and set of tokens in sequence 2.

```
Source:
c b a a b <s> c b a e f

Target:
c b a e f
```

Input Generation Sample max length l, then sample partition of l to l_1 and l_2, which will be the lengths of sequence 1 and sequence 2. Sample $l/2$ unique tokens. To get sequence 1, sample l_1 of the unique tokens. To get sequence 2, sample l_2 of the unique tokens.

G.6 Set1 Minus Set2

Definition Input is token sequence 1 and token sequence 2. Output is a token sequence that is the subtraction of the set of tokens in sequence 2 from the set of tokens in sequence 1.

```
Source:
c b a a b <s> c a e f

Target:
b
```

Input Generation Sample max length l, then sample partition of l to l_1 and l_2, which will be the lengths of sequence 1 and sequence 2. Sample $l/2$ unique tokens. To get sequence 1, sample l_1 of the unique tokens. To get sequence 2, sample l_2 of the unique tokens.

G.7 Replace

Definition Input is token sequence 1 and token sequence 2. Token sequence 2 always has two tokens, which we will denote as t_1 and t_2. Output is token sequence 1 with t_1 replaced by t_2.

```
Source:
c b a a b <s> a x

Target:
c b x x b
```

Input Generation To get sequence 1, sample input sequence length l uniformly between min and max possible length, then sample l tokens. To get sequence 2, sample one token from the set of tokens in sequence 1, and sample a token from the entire vocab.

G.8 Duplicate

Definition Input is a token sequence. Output is the input token sequence with every token duplicated.

```
Source:
c a b a a b

Target:
c c a a b b a a a a b b
```

Input Generation Sample input sequence length l uniformly between min and max possible length, then sample l tokens.

G.9 Intersect

Definition Input is token sequence 1 and token sequence 2. Output is token sequence that is the intersection of the set of tokens in sequence 1 and set of tokens in sequence 2.

```
Source:
c a b c a a b <s> a a b

Target:
a b
```

Input Generation Sample max length l, then sample partition of l to l_1 and l_2, which will be the lengths of sequence 1 and sequence 2. Sample $l/2$ unique tokens. To get sequence 1, sample l_1 of the unique tokens. To get sequence 2, sample l_2 of the unique tokens.

G.10 Reverse

Definition Input is a token sequence. Output is the input token sequence in reverse order.

```
Source:
c a b c a a b

Target:
b a a c b a c
```

Input Generation Sample input sequence length l uniformly between min and max possible length, then sample l tokens.

G.11 Set2 Minus Set1

Definition Input is token sequence 1 and token sequence 2. Output is a token sequence that is the subtraction of the set of tokens in sequence 1 from the set of tokens in sequence 2.

```
Source:
c b c b b <s> a a b x y z

Target:
a x y z
```

Input Generation Sample max length l, then sample partition of l to l_1 and l_2, which will be the lengths of sequence 1 and sequence 2. Sample $l/2$ unique tokens. To get sequence 1, sample l_1 of the unique tokens. To get sequence 2, sample l_2 of the unique tokens.

G.12 Deduplicate

Definition Input is a token sequence. Output is the input token sequence with no adjacent duplicate tokens.

```
Source:
c b b c c d d d d e f e

Target:
c b c d e f e
```

Input Generation Uniformly sample the length l of the input from min length to max length, then sample t tokens that will be in the input where $t < l$, then sample the amount of each token by randomly partitioning l to t parts $\{p_1, p_2 ... p_t\}$, then generate the input sequence by iterating through the t tokens and repeating each one p_i times.

G.13 Last Token

Definition Input is a token sequence. Output is the last token in the input sequence.

```
Source:
c b b c c a

Target:
a
```

Input Generation Sample input sequence length l uniformly between min and max possible length, then sample l tokens.

G.14 First Token

Definition Input is a token sequence. Output is the first token in the input sequence.

```
Source:
c b b c c a

Target:
c
```

Input Generation Sample input sequence length l uniformly between min and max possible length, then sample l tokens.

G.15 Longest Word

Definition Input is a token sequence. The 'input token sequence has a special format, with special separator tokens distributed throughout. Output is the largest number of contiguous tokens.

```
Source:
c b <s> s <s> b a f <s> s d <s> a a b c w q
Target:
6
```

Input Generation Sample the number of "words" w defined as a contiguous block of normal tokens. Sample the length of the input token sequence l. Partition $l - w + 1$ to w parts $p_1, p_2, ...p_w$. Build the input sequence by sampling the "words" according to the word lengths $p_1, p_2, ...p_w$, and then joining the words with the special separator token.

G.16 Search

Definition Input is token sequence 1 and token sequence 2. Output is "yes" if token sequence 2 is contained in token sequence 1, and "no" otherwise.

```
Source:
a a b c w q a b d e <s> c w e

Target:
no
```

Input Generation With 0.75 probability, generate sequence 1 and sequence 2 such that sequence 2 appears in sequence 1 at least once by using the following procedure: sample sequence 2, sample sequence 1 length, sample location that sequence 2 appears in sequence 1, sample remaining tokens in sequence 1. With 0.25 probability, generate sequence 1 and sequence 2 such that sequence 2 does not appear in sequence 1 by using the following procedure: sample sequence 1, sample sequence 2 and keep re-sampling sequence 2 until it does not appear in sequence 1.

G.17 Length

Definition Input is a token sequence. Output is the number of tokens in the input token sequence.

```
Source:
a a b c w

Target:
5
```

Input Generation Sample input sequence length l uniformly between min and max possible length, then sample l tokens.

G.18 Count

Definition Input is token sequence 1 and token sequence 2. Token sequence 2 is only 1 token which we will denote t. Output is the number of times t occurs in token sequence 1.

Example

```
Source:
a a b c a a a w <s> a

Target:
5
```

Input Generation Start by sampling the single token t in sequence 2. Then sample l, which will be the length of sequence 1, and sample the number of times c that t will appear in token 1. Sample $l - c$ remaining tokens.

H More Initialization Schemes

We define two other initialization schemes we tried, and show results in Appendix Table 5

Across Layers Scale We tried the `Across_Layers_Scale` initialization scheme, where we initialized each *type* of parameter with the scale statistic of it computed across all layers. Some examples of types of parameters include query matrices, pre-MLP layer norms, and pre-attention layer norms. Using this initialization, all query matrices across the whole model would be initialized with a single SD, which is the SD over all the pre-trained Q matrix weights, across all layers. We find that `Across_Layers_Scale` provides significant benefits for `LIME`, `Identity`, and `Set`, preserving at least 87% of the Per Param Mean/SD initialization scheme benefit for CNNDM-10K and MTOP. This result suggests that parameter statistics varying from layer to layer might not be an important factor for synthetic pre-training initialization benefits.

Per Layer Scale We tried the `Per_Layer_Scale` initialization scheme, where every layer is initialized with just two statistics: the SD of all the non-layer norm parameter weights in that layer and the mean of all the layer norm parameter weights in that layer. Results are mixed. There is no benefit for `Identity` CNNDM-10K and minimal benefit for `LIME` MTOP. For the other four pre-training and downstream task settings, there was significant benefit, preserving at least 80% of the Per Param Mean/SD benefit for CNNDM-10K and MTOP.

I Compute Used

Each experiment was ran on a Google Cloud v3-8 TPU VM. We ran most experiments using a total of 5 such VMs.

J Additional Experiments

Multiple Seeds for Section 3 and 4 Experiments In Appendix Table 6, we show the mean and standard deviation for five seeds from evaluating Pre-trained T5, Random Init, LIME, and Set on the six downstream tasks. For LIME and Set, for each different seed, we generated and pre-trained on a new synthetic dataset.

More Tasks for Section 5.1 Per Param Mean/SD Initialization In Appendix Table 7, we show results for the Section 5.1 Per Param Mean/SD init scheme for all six tasks, not just CNNDM-10K and MTOP which were the only two tasks shown in Table 2.

Multiple Seeds for Section 5.1 Per Param Mean/SD Initialization On CNNDM-10K, MTOP, and SQuAD, we run the Per Param Mean/SD ininitialization experiments for five seeds. Results are shown in Appendix Table 7. We initialized using a model pre-trained on a different generated synthetic dataset for each seed.

More Tasks for Section 5.3 Pre-attention Layer Norm Initialization Sweep In Appendix Table 8, we show results for the Section 5.3 pre-attention layer norm initialization sweep for all six tasks, not just CNNDM-10K and MTOP which were the only two tasks shown in Table 3.

Larger Model Size In Appendix Table 9, we run experiments with T5 Base, which is a larger model size than T5 Small that was used for experiments in the main paper. T5 Base has 220 million parameters with 12 encoder/decoder layers compared to 60 million parameters with 6 encoder/decoder layers for T5 Small. `LIME`, `Set`, `LIME Per_Param_Mean_SD`, and `Set Per_Param_Mean_SD` close a large proportion of the gap to natural pre-training, similar to results for the T5 Small experiments.

Table 4: Dataset size ablation. Retrosynthesis-40K and SQuAD-87K are the same as Retrosynthesis and SQuAD in Table 1. The CNNDM-10K and CNNDM-100K results as well as the Retrosynthesis-10K and Reotrosynthesis-40K results show that benefits from synthetic (and T5 natural language) pre-training increase as the amount of training data for a task decrease.

	Retrosyn-1K	Retrosyn-10K	Retrosyn-40K
Pre-trained T5	7.6	29.5	43.1
LIME	4.0	25.1	41.1
Random Init	0.0	18	39.2

	CNNDM-1K	CNNDM-10K	CNNDM-100K
Pre-trained T5	29.6	35.8	39.1
LIME	28.4	33.2	37.6
Random Init	12.2	18.9	35.1

	SQuAD-1K	SQuAD-10K	SQuAD-87K
Pre-trained T5	19.5/36.8	59.2/73.5	77.5/86.0
LIME	3.7/8.5	23.21/34.1	50.4/62.1
Random Init	0.8/0.9	0.3/1.6	17.2/25.2

Table 5: Evaluating different initialization schemes with statistics of LIME, Set, and Identity pre-trained models.

	CNNDM-10K			MTOP		
	LIME	Set	Identity	LIME	Set	Identity
Section 5.1						
Full Init	33.2	32.8	30.2	73.7/94.0	71.7/93.7	68.6/93.0
Per Param Mean/SD	29.3	29.8	28.8	66.1/92.3	66.6/92.3	67.8/92.6
Section 5.2						
Per Param Scale	29.0	29.5	29.0	67.3/92.5	66.5/92.4	66.3/92.3
Across Layers Scale	27.8	28.8	28.5	63.5/91.5	66.9/92.5	67.5/92.6
Per Layer Scale	27.1	28.1	19.6	48.1/85.3	61.1/90.5	60.9/90.5
Whole Model Scale	15.5	26.1	20.5	34.3/81.4	38.7/83.1	59.8/91.0
Pre-attn LN Per Param Scale	27.8	25.1	27.3	59.9/90.3	58.9/89.0	46.7/86.3
Attention Per Param Scale	17.2	19.1	19.1	19.3/73.9	41.5/84.9	38.4/82.9
Pre-MLP LN Per Param Scale	20.2	20.6	20.5	41.8/85.1	40.3/84.5	43.0/85.4
MLP Per Param Scale	16.6	18.7	18.3	36.5/80.8	36.5/81.6	37.9/83.0

Baseline	CNNDM-10K	MTOP
Random Init	18.9	38.6/83.0
Pre-trained T5	35.8	81.0/95.2

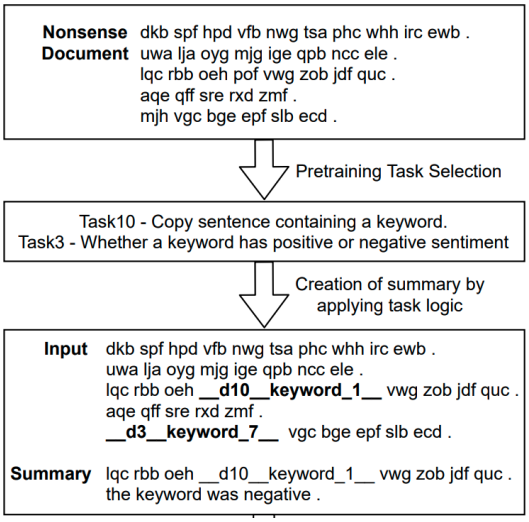

Figure 5: Example of a Nonsense Summary task data point. Figure is directly taken from (Krishna et al., 2021).

Table 6: Same table as Table 1 in main paper except with mean and standard deviation over 5 seeds for Pre-trained T5, Random Init, LIME, and Set. For LIME and Set, for each seed, a different synthetic pre-training set was generated and pre-trained on.

	CNNDM-10K ROUGE1	MTOP EM/F1	WebQSP EM/F1	SQuAD EM/F1	Code Trans. EM	Retrosyn. EM
			Baselines			
Pre-trained T5	**35.8 (0.1)**	**81.0/95.2 (0.3/2.2)**	**82.3/91.7 (0.8/0.4)**	**77.4/86.1 (0.2/0.2)**	**61.2 (0.7)**	**43.5 (0.3)**
Wiki 10K	34.0	71.6/94.2	79.6/90.6	67.1/76.9	60.2	41.1
Random Init	18.9 (0.5)	38.6/83.0 (2.1/1.2)	25.8/71.5 (0.7/0.5)	16.0/24.2 (2.1/2.0)	57.6 (0.3)	39.0 (0.5)
			Section 3.1: Previously Proposed Synthetic Tasks			
LIME	**33.2 (0.7)**	**73.7/94.0 (0.8/0.1)**	**76.2/89.6 (1.2/0.6)**	**49.6/61.1 (1.6/1.3)**	**58.3 (1.1)**	**40.7 (0.4)**
Dyck	27.1	65.9/91.9	58.5/83.5	50.3/62.4	58.8	40.4
Nons. Summary	32.0	68.0/92.7	65.2/85.2	48.4/60.1	57.3	39.6
			Section 4.1: Simpler Synthetic Tasks			
Set	**32.8 (0.5)**	**71.7/93.7 (0.9/0.2)**	**73.6/88.6 (1.0/0.3)**	**48.2/60.0 (0.5/0.7)**	**58.8 (0.8)**	**40.5 (0.9)**
Identity	30.2	68.6/93.0	69.8/86.8	26.1/35.4	57.8	40.5

Table 7: Section 5.1 Per Param Mean/SD initialization results for all downstream tasks (rather than just CNNDM-10K and MTOP which were the only two shown in Table 2). Baseline results and normal fune-tune results are from Appendix Table 6. For CNNDM-10K, MTOP, and SQuAD we provide the mean/SD for five seeds, where for each seed we ran the initialization with a different pre-trained model with different generated data. Per Param Mean/SD initialization gave CNNDM-10K, MTOP, SQuAD, and WebQSP noticeable gains over Random Init, close to gains from Full Init synthetic pre-trained models.

	CNNDM-10K			MTOP		
	LIME	Set	Identity	LIME	Set	Identity
Full Init	33.2 (0.7)	32.8 (0.5)	30.2	73.7/94.0 (0.8/0.1)	71.7/93.7 (0.9/0.2)	68.6/93.0
Per Param Mean/SD	28.7 (0.7)	29.4 (0.6)	29.2 (0.6)	67.9/92.6 (3.3/0.8)	67.3/92.7 (2.5/0.6)	66.6/92.2 (1.4/0.5)

Baseline	CNNDM-10K	MTOP
Random Init	18.9 (0.5)	38.6/83.0 (2.1/1.2)
Pre-trained T5	35.8 (0.1)	81.0/95.2 (0.3/2.2)

	SQuAD			WebQSP		
	LIME	Set	Identity	LIME	Set	Identity
Full Init	49.6/61.1 (1.6/1.3)	48.2/60.0 (0.5/0.7)	26.1/35.4	76.2/89.6 (1.2/0.6)	73.6/88.6 (1.0/0.3)	69.8/86.8
Per Param Mean/SD	35.4/45.5 (9.2/10.4)	31.1/40.4 (6.5/7.4)	25.6/34.4 (4.9/5.7)	60.9/84.8	63.0/86.1	58.9/83.6

Baseline	SQuAD	WebQSP
Random Init	16.0/24.2 (2.1/2.0)	25.8/71.5 (0.7/0.5)
Pre-trained T5	77.4/86.1 (0.2/0.2)	82.3/91.7 (0.8/0.4)

	Code Trans.			Retrosyn.		
	LIME	Set	Identity	LIME	Set	Identity
Full Init	58.3 (1.1)	58.8 (0.8)	57.8	40.7 (0.4)	40.5 (0.9)	40.5
Per Param Mean/SD	58.0	58.4	57.2	40.3	38.7	39.3

Baseline	Code Trans.	Retrosyn.
Random Init	57.6 (0.3)	39.0 (0.5)
Pre-trained T5	61.2 (0.7)	43.5 (0.3)

Table 8: Pre-attention layer norm initialization sweep results for all downstream tasks (rather than just CNNDM-10K and MTOP which were the only two shown in Table 3). Baseline results are from Appendix Table 6. Lower pre-attention layer norm value gave noticeable benefits over Random Init for CNNDM-10K, MTOP, WebQSP, and SQuAD.

Pre-attn LN Init Value	CNNDM-10K	MTOP	WebQSP	SQuAD	Code Trans.	Retrosyn.
0.05	28.2	25.7/73.3	30.5/71.2	22.8/31.2	57.9	39.3
0.1	28.3	32.5/77.5	28.9/70.6	23.5/31.7	57.6	39.9
0.2	28.2	56.4/89.3	30.6/72.3	28.2/37.3	57.0	39.4
0.4	24.4	56.9/89.9	30.0/72.8	36.3/46.9	57.3	39.6
0.8	18.6	50.7/88.5	28.0/71.6	34.2/44.6	57.2	39.9

Baseline	CNNDM-10K	MTOP	WebQSP	SQuAD	Code Trans.	Retrosyn.
Random Init	18.9 (0.5)	38.6/83.0 (2.1/1.2)	25.8/71.5 (0.7/0.5)	16.0/24.2 (2.1/2.0)	57.6 (0.3)	39.0 (0.5)
Pre-trained T5	35.8 (0.1)	81.0/95.2 (0.3/2.2)	82.3/91.7 (0.8/0.4)	77.4/86.1 (0.2/0.2)	61.2 (0.7)	43.5 (0.3)

K T5 Small Parameters List

```
decoder/decoder_norm/scale
decoder/layers_0/encoder_decoder_attention/key/kernel
decoder/layers_0/encoder_decoder_attention/out/kernel
decoder/layers_0/encoder_decoder_attention/query/kernel
decoder/layers_0/encoder_decoder_attention/value/kernel
decoder/layers_0/mlp/wi/kernel
decoder/layers_0/mlp/wo/kernel
decoder/layers_0/pre_cross_attention_layer_norm/scale
decoder/layers_0/pre_mlp_layer_norm/scale
decoder/layers_0/pre_self_attention_layer_norm/scale
decoder/layers_0/self_attention/key/kernel
decoder/layers_0/self_attention/out/kernel
decoder/layers_0/self_attention/query/kernel
decoder/layers_0/self_attention/value/kernel
decoder/layers_1/encoder_decoder_attention/key/kernel
decoder/layers_1/encoder_decoder_attention/out/kernel
decoder/layers_1/encoder_decoder_attention/query/kernel
decoder/layers_1/encoder_decoder_attention/value/kernel
decoder/layers_1/mlp/wi/kernel
decoder/layers_1/mlp/wo/kernel
decoder/layers_1/pre_cross_attention_layer_norm/scale
decoder/layers_1/pre_mlp_layer_norm/scale
decoder/layers_1/pre_self_attention_layer_norm/scale
decoder/layers_1/self_attention/key/kernel
decoder/layers_1/self_attention/out/kernel
decoder/layers_1/self_attention/query/kernel
decoder/layers_1/self_attention/value/kernel
decoder/layers_2/encoder_decoder_attention/key/kernel
decoder/layers_2/encoder_decoder_attention/out/kernel
decoder/layers_2/encoder_decoder_attention/query/kernel
decoder/layers_2/encoder_decoder_attention/value/kernel
decoder/layers_2/mlp/wi/kernel
decoder/layers_2/mlp/wo/kernel
decoder/layers_2/pre_cross_attention_layer_norm/scale
decoder/layers_2/pre_mlp_layer_norm/scale
decoder/layers_2/pre_self_attention_layer_norm/scale
decoder/layers_2/self_attention/key/kernel
decoder/layers_2/self_attention/out/kernel
decoder/layers_2/self_attention/query/kernel
decoder/layers_2/self_attention/value/kernel
decoder/layers_3/encoder_decoder_attention/key/kernel
decoder/layers_3/encoder_decoder_attention/out/kernel
decoder/layers_3/encoder_decoder_attention/query/kernel
decoder/layers_3/encoder_decoder_attention/value/kernel
decoder/layers_3/mlp/wi/kernel
decoder/layers_3/mlp/wo/kernel
decoder/layers_3/pre_cross_attention_layer_norm/scale
decoder/layers_3/pre_mlp_layer_norm/scale
decoder/layers_3/pre_self_attention_layer_norm/scale
decoder/layers_3/self_attention/key/kernel
decoder/layers_3/self_attention/out/kernel
decoder/layers_3/self_attention/query/kernel
decoder/layers_3/self_attention/value/kernel
decoder/layers_4/encoder_decoder_attention/key/kernel
decoder/layers_4/encoder_decoder_attention/out/kernel
decoder/layers_4/encoder_decoder_attention/query/kernel
decoder/layers_4/encoder_decoder_attention/value/kernel
decoder/layers_4/mlp/wi/kernel
decoder/layers_4/mlp/wo/kernel
decoder/layers_4/pre_cross_attention_layer_norm/scale
decoder/layers_4/pre_mlp_layer_norm/scale
decoder/layers_4/pre_self_attention_layer_norm/scale
decoder/layers_4/self_attention/key/kernel
decoder/layers_4/self_attention/out/kernel
decoder/layers_4/self_attention/query/kernel
decoder/layers_4/self_attention/value/kernel
decoder/layers_5/encoder_decoder_attention/key/kernel
decoder/layers_5/encoder_decoder_attention/out/kernel
decoder/layers_5/encoder_decoder_attention/query/kernel
decoder/layers_5/encoder_decoder_attention/value/kernel
decoder/layers_5/mlp/wi/kernel
decoder/layers_5/mlp/wo/kernel
decoder/layers_5/pre_cross_attention_layer_norm/scale
decoder/layers_5/pre_mlp_layer_norm/scale
decoder/layers_5/pre_self_attention_layer_norm/scale
decoder/layers_5/self_attention/key/kernel
decoder/layers_5/self_attention/out/kernel
decoder/layers_5/self_attention/query/kernel
```

```
decoder/layers_5/self_attention/value/kernel
decoder/relpos_bias/rel_embedding
encoder/encoder_norm/scale
encoder/layers_0/attention/key/kernel
encoder/layers_0/attention/out/kernel
encoder/layers_0/attention/query/kernel
encoder/layers_0/attention/value/kernel
encoder/layers_0/mlp/wi/kernel
encoder/layers_0/mlp/wo/kernel
encoder/layers_0/pre_attention_layer_norm/scale
encoder/layers_0/pre_mlp_layer_norm/scale
encoder/layers_1/attention/key/kernel
encoder/layers_1/attention/out/kernel
encoder/layers_1/attention/query/kernel
encoder/layers_1/attention/value/kernel
encoder/layers_1/mlp/wi/kernel
encoder/layers_1/mlp/wo/kernel
encoder/layers_1/pre_attention_layer_norm/scale
encoder/layers_1/pre_mlp_layer_norm/scale
encoder/layers_2/attention/key/kernel
encoder/layers_2/attention/out/kernel
encoder/layers_2/attention/query/kernel
encoder/layers_2/attention/value/kernel
encoder/layers_2/mlp/wi/kernel
encoder/layers_2/mlp/wo/kernel
encoder/layers_2/pre_attention_layer_norm/scale
encoder/layers_2/pre_mlp_layer_norm/scale
encoder/layers_3/attention/key/kernel
encoder/layers_3/attention/out/kernel
encoder/layers_3/attention/query/kernel
encoder/layers_3/attention/value/kernel
encoder/layers_3/mlp/wi/kernel
encoder/layers_3/mlp/wo/kernel
encoder/layers_3/pre_attention_layer_norm/scale
encoder/layers_3/pre_mlp_layer_norm/scale
encoder/layers_4/attention/key/kernel
encoder/layers_4/attention/out/kernel
encoder/layers_4/attention/query/kernel
encoder/layers_4/attention/value/kernel
encoder/layers_4/mlp/wi/kernel
encoder/layers_4/mlp/wo/kernel
encoder/layers_4/pre_attention_layer_norm/scale
encoder/layers_4/pre_mlp_layer_norm/scale
encoder/layers_5/attention/key/kernel
encoder/layers_5/attention/out/kernel
encoder/layers_5/attention/query/kernel
encoder/layers_5/attention/value/kernel
encoder/layers_5/mlp/wi/kernel
encoder/layers_5/mlp/wo/kernel
encoder/layers_5/pre_attention_layer_norm/scale
encoder/layers_5/pre_mlp_layer_norm/scale
encoder/relpos_bias/rel_embedding
```

Table 9: We run experiments with T5 Base, which is a larger model size than T5 Small that was used for experiments in the main paper. T5 Base has 220 million parameters with 12 encoder/decoder layers compared to 60 million parameters with 6 encoder/decoder layers for T5 Small. `LIME`, `Set`, `LIME` `Per_Param_Mean_SD`, and `Set` `Per_Param_Mean_SD` close a large proportion of the gap to natural pre-training, similar to results for the T5 Small experiments.

Experiments With Larger Model Size (T5 Base)

	CNNDM-10K ROUGE 1	MTOP EM/F1	SQuAD EM/F1	Retrosynthesis EM	Average	
Pre-trained_T5	37.6	83.8/96.8	83.6/91.1	44.4	62.4	100.0%
Random_Init	16.4	30.30/78.3	7.5/9.9	36.9	22.8	0.0%
LIME	33.7	72.5/93.8	48.4/60.4	38.2	48.2	57.9%
Set	34.4	74.4/94.2	51.8/62.7	39.6	50.1	65.4%
LIME Per_Param_Mean_SD	26.5	59.2/90.5	25.9/34.2	37.3	37.2	32.8%
Set Per_Param_Mean_SD	27.4	64.9/92.2	22.4/30.6	37.5	38.1	36.0%

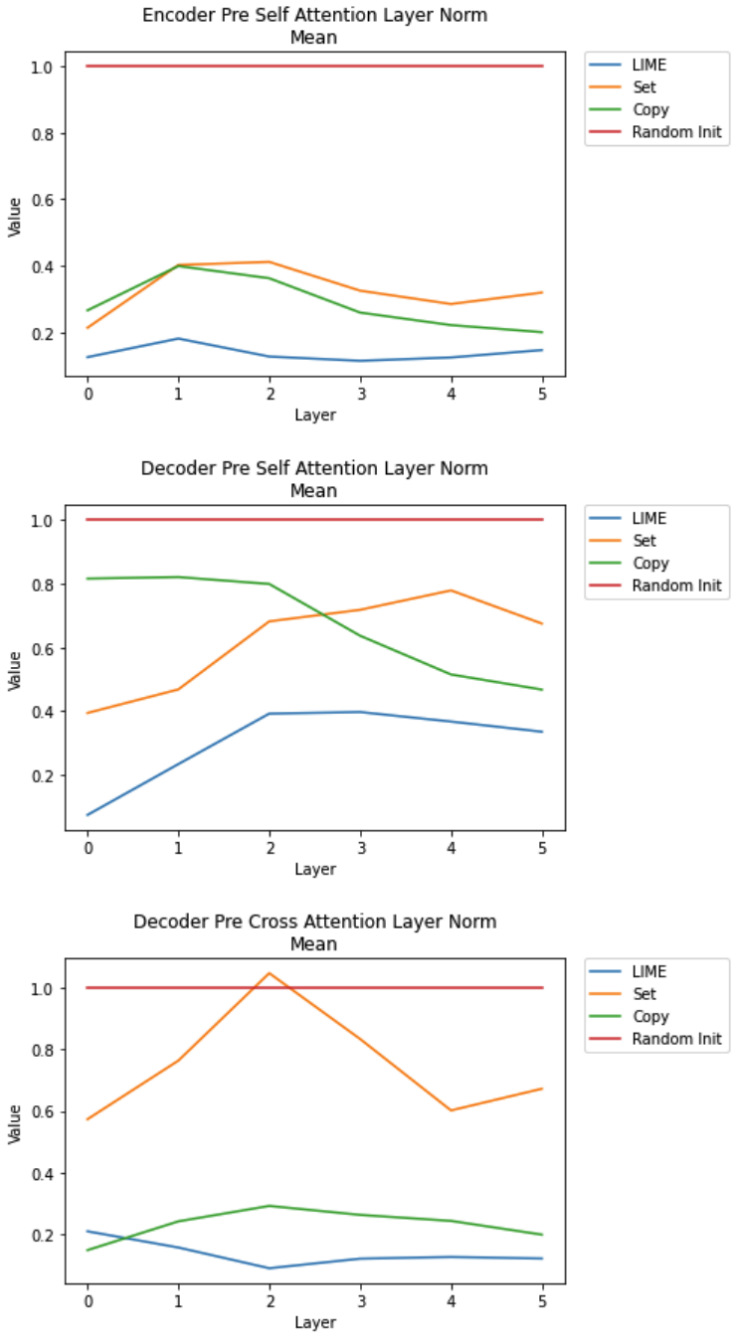

Figure 6: For each layer, compute the mean over weights of the pre-attention layer norm parameter in that layer. Compute and plot these values for a Random Init model as well as LIME, Set, and Identity pre-trained models. For example, looking at the top-most plot, the mean of the Set pre-trained model's encoder layer 0 pre-attention layer norm is about 0.2. And the mean of the Identity pre-trained model's encoder layer 1 pre-attention layer norm is about 0.4. Looking at the bottom-most plot, the mean of the Set pre-trained model's encoder layer 4 pre-cross-attention layer norm is about 0.6.