Tackling BabyLM Challenge Using Transformer Based Approaches

Can Gözpınar
cgozpinar18@ku.edu.tr

Aydin Ahmadi
aahmadi22@ku.edu.tr

1 Problem statement

The BabyLM challenge involves the development of an effective language model which can be tested on the classification task proposed in evaluation pipeline of the challenge. The dataset contains spoken and written language acquired by children during their early years. Several significant hurdles need to be addressed to tackle this challenge successfully.

To begin with, the dataset is relatively small, posing difficulties in training a language model with sufficient accuracy and generalization. Furthermore, the tokenizer that will be trained on the dataset will have a signifcant amount of tokens which are specific to some of files (for instance in aochiles file) and are not repeated in other files. Moreover, the dataset is organized according to the child's language acquisition process, ranging from words heard by infants under one year of age to more complex language used by teenagers. This presents a unique challenge in modeling the progression of language skills accurately. Additionally, the dataset exhibits significant variations in sentence and passage lengths, demanding careful consideration in designing a tokenizer and model architecture that can handle such variability effectively.

Overcoming these challenges in developing a language model for a limited corpus can have substantial benefits for the natural language processing and cognitive science communities. Our project aims to explore the effects of various curriculum learning based approaches for training models to address these challenges.

2 What you proposed vs. what you accomplished

What we proposed in our project proposal:

- In our proposal, we planned to explore *T-Fixup*. We did not do this since this was no longer a good fit once we had to scale down our transformer models and their training epochs. Since we were training for shorter epochs than what we anticipated during the writing of our proposal, we did not have stability issues during training.
- We started with *T5* baseline provided but abandoned it later and developed our set of smaller model baselines. We made this switch since we could not tolerate the long training hours these provided baseline models required.
- ~~We implemented a curriculum learning like approach in which we gradually trained our model on harder tasks.~~
- ~~We planned to use the evaluation pipeline provided by the BabyLM challange. Our code implementation supports this pipeline and we have used it for the evaluation of certain experiments.~~

What we added on top of our project proposal:

- We experimented with models at different scales.
- Experimented models from different transformer families (Encoder: BERT, Decoder: GPT2).
- Experimented with more than one curriculum learning based approach.
- Implemented a much more comprehensive and general purpose code base than we were set out to do. It can support a diverse set of experiments without even needing to change it. These experiments do not even need to be related to our proposed approach.

3 Related work

The core of the text generation task is to model a mapping from input to output. The inputs can have various forms like graphs, tables, or multimedia. The outputs are the varied-size of the sequences of a text. The BabyLM challenge considers the inputs as sequences of words and outputs a sequence which is generated by a language model.

By the advent of transformer architecture (Vaswani et al., 2017), using pretrained language models (PLM) have gained the attention of researchers. The idea behind the PLM's is to train the models in large-scale corpus and then fine-tune them in a downstream task. These models can develop a universal understanding of the language. This can be beneficial when a pretrained model is fine-tuned on a downstream task in comparison to training a model on the specific task from scratch. Work of (Brown et al., 2020) is an example of this phenomena.

There exists different families of transformer architectures. These pretrained models can utilize different pretraining objectives according to the architecture they adapt. For example, architectures like T5 (Raffel et al., 2020) and Bart (Lewis et al., 2020) use a standard encoder-decoder transformer, whereas models like GPT (Brown et al., 2020) utilize decoder-only architecture, and BERT (Devlin et al., 2019) encoder-only. Different architectures seem to strive at different tasks. For instance (Raffel et al., 2020) have arrived at the conclusion that using the encoder-decoder architecture is more beneficial. The following discussion present some recent research on using different methodologies that aims to have meaningfully generated text by leveraging the pretrained language models.

(Lee et al., 2020) have presented a work utilizing a transformer-based sentence-level tokenization model. They added trainable sentence embeddings indicating the index of each sentence to each of the word representations. They trained the sentence embedding on a BERT to have the encoded representations. Then, they used a sequence reconstructor which consists of a decoder and a pointer network to predict the sequence order after shuffling input sentences. Their results show that their model is comparable to T5.

(Araujo et al., 2021) presented a work that similarly uses sentence-level representations. They got inspiration from predictive coding (van Berkum et al., 2005) which is a neuroscience theory on language development. However, they predict the next future sentence on a top-down pathway with a next-sentence loss + BERT loss. They discuss that their model learns discourse-level representations and sentence relationships.

(Ji and Huang, 2021) introduces a novel latent variable model that learns a sequence of discrete latent variables from long text and utilizes them to guide the text generation process, ensuring coherence in the output. This allows the model to abstract the discourse structure of the text and generate coherent long texts with interpretable latent codes.

(Lee et al., 2022) presents a concept-based curriculum learning approach to language modeling. They work a masking mechanism in a curriculum that begins with the core concepts in human language acquisition and ends with complex ones. In their approach, They first identify some of the core concepts in words and phrases to be masked by scoring concepts in a ConceptNet in a top-down manner. They gradually mask concepts related to the previous masked concepts in the consecutive stages. They show the effectiveness of their algorithms for masking in masked language models in their paper.

In (Hacohen and Weinshall, 2019) they presented a curriculum learning approach which claimed to improve the training of deep neural networks. This paper specifically introduced two functions in order to train the neural networks and focused on CNN's. The first one is a pacing function which controls the pace or progression of difficulty in the mini-batches of training examples. The second one is a bootstrapping function which scores the difficulty o batches while training by sorting their losses. By this method they have re-trained their model to get better results

4 Your dataset

We used the dataset provided for the STRICT SMALL track of the BabyLM challange 1. It is comprised of data files which are already assigned to training, validation and testing splits. These data files and their characteristics is listed below. A further investigation based on our observations can be found in 6.0.2.

- *aochildes*: The data has been taken from BabyBERTa's (Huebner et al., 2021) dataset.

This file contains words of American-English child-directed speech.

- *switchboard*: The data has been collected from Switchboard dialogue act corpus which contains dialogues.
- *bnc_spoken*: The data has been collected from Oxford's Literary and Linguistic Data Service which consists of long dialogues and presentations.
- *cbt*: The data consists of children book sentences.
- *children_stories*: The data consists of children book sentences.
- *gutenberg*: the data has been taken from Standardized Project Gutenberg Corpus.
- *simple_wikipedia*: The data is scraped from wikipedia pages which their words may seen familiar and simpler to a child.
- *open_subtitles*: The data contains subtitiles from movies. While preprocessing duplicate documents has been removed from the scraped ones.
- *qed*: The data is an open multilingual collection of subtitles for educational videos and lectures collaboratively transcribed and translated over the AMARA web-based platform.
- *wikipedia*: The data is scraped from wikipedia pages and contains complicated english words.

4.1 Data preprocessing

The initial preprocessing of the data is provided by the preprocessing pipeline of the BabyLM challange. We have downloaded the data files which were intially preprocessed using this pipeline and built our work no top of it. Briefly, the inner workings of this pipeline is as follows. As the first step of data collection, txt files are scraped from the targeted web pages and books. The texts are being scraped from the appointed section of books or webpages. In the second step, they sample from these txt files and assign them to training, validation and testing splits. In the context of training models based on huggingface Transformers library, preprocessing steps are implemented to ensure that the data is appropriately formatted and ready for input into the language model.

Firstly, the dataset loading step involves retrieving the train, validation, and test datasets from specific directories. This step allows for flexibility in loading specific dataset files by specifying their names or loading all available data files. By leveraging the Hugging Face library's functions.

Next, the tokenization step is applied to the raw dataset. In this case, HuggingFace's Tokenizers library is utilized. The tokenization process involves segmenting the text into individual tokens, truncating sequences that exceed the tokenizer's maximum length, and returning the tokenized dataset.

Following tokenization, the data collation step is performed. Two separate data collator functions are defined, each catering to a specific pretraining task: causal language modeling (CLM) and masked language modeling (MLM). The data collator functions handle important tasks such as padding the sequences to ensure consistent sequence lengths, generating attention masks, and creating labels for the language modeling tasks. For CLM, the collator pads the inputs and generates labels by duplicating the input sequences. Another processing is required to shift the labels by one token to the left in order to have the next tokens in the input as the labels. This processing is left to the implementation of the model. We left it this way and designed our custom models compliant to this scheme in order to keep our code compatible with HuggingFace's API in general. Furthermore the Data Collator for the Masked Language Modeling (mlm) pretraining objective pads the inputs, and randomly performs perturbations on the input sequence. In our implementation, these perturbations on the tokens can be masking, replacing with a random token. Their probabilities are as follows, 80% masked, $10 + \%$ random, and 10% original. It is important to note that padding the sequences are done by the data collators. They are padding the sequences in a batch to the length of the longest sequence in the given batch. As a result, different batches can have different padded sequence lengths. We are using padding tokens defined in our tokenizer to pad these sequences and map them to an index(*we used: ignore_index* $= -100$ *as by default in torch*) which will later be discarded by our loss functions so that they will not contribute to the computation of the gradients during the model optimization phase. In a simi-

Dataset	Domain	# Words		Proportion
		STRICT-SMALL	STRICT	
CHILDES (MacWhinney, 2000)	Child-directed speech	0.44M	4.21M	5%
British National Corpus (BNC),[1] dialogue portion	Dialogue	0.86M	8.16M	8%
Children's Book Test (Hill et al., 2016)	Children's books	0.57M	5.55M	6%
Children's Stories Text Corpus[2]	Children's books	0.34M	3.22M	3%
Standardized Project Gutenberg Corpus (Gerlach and Font-Clos, 2018)	Written English	0.99M	9.46M	10%
OpenSubtitles (Lison and Tiedemann, 2016)	Movie subtitles	3.09M	31.28M	31%
QCRI Educational Domain Corpus (QED; Abdelali et al., 2014)	Educational video subtitles	1.04M	10.24M	11%
Wikipedia[3]	Wikipedia (English)	0.99M	10.08M	10%
Simple Wikipedia[4]	Wikipedia (Simple English)	1.52M	14.66M	15%
Switchboard Dialog Act Corpus (Stolcke et al., 2000)	Dialogue	0.12M	1.18M	1%
Total	–	9.96M	98.04M	100%

Figure 1: The datasets for the STRICT and STRICT-SMALL tracks of the BabyLM Challenge. (Warstadt et al., 2023)

lar approach, the data collator for the masked language modeling would generate labels for the non masked or randomly replaced tokens a label which will be ignored by the loss function. This stems from the fact that, in this scheme, our model's objective is to predict the changed tokens and not the already available tokens which would be a simple copy operation on the input token to predict the correct output which would not be a helpful task for pretraining our models.

4.2 Data annotation

The STRICT SMALL challange of BabyLM prohibits the use of data other than what was provided and aims to develop a successfull pretraining scheme. Since pretraining schemes are mostly self-supervised, and the use of external data was not permitted, we did not annotate any data.

5 Baselines

5.1 Baselines Provided by the BabyLM Challange

Some naive baselines have been provided by the BabyLM challenge for the Strict-Small track which we are working on. They have implemented OPT-125m, RoBERTa-base, T5-base models as baslines for the challenge. OPT-125 is a decoder-only language model which is trained by casual language modeling task. It uses BPE as its tokenizer, and is a large 125 million parameter model.T5-base is a encoder-decoder multi-task model which can be both trained by casual language modeling task or masked language modeling task. It uses SentencePiece as its tokenizer and is a smaller parameterd version on T5. RoBERTa-base is a encoder-only classification model which is trained by masked language modeling task. It uses BPE as its tokenizer and is an approximately 125 million parametered model. The evaluation results reported for these baselines are provided in 1. Their corresponding parameters are reported in 2. Initially, we have also implemented initialized a T5 model and trained it on our dataset from scratch to use it as a baseline. It used a SentencePiece tokenizer with the vocabulary size of 30000 and it was trained on the whole dataset. The evaluation results for this model is reported in 3. Unfortunately, due to the cost of training this model we had to cut its training short and opt for different solutions which we explore in 5.2.

metric	T5-base	Roberta-base	OPT-125m
Anaphor Agr.	68.9	81.5	63.8
Agr. Structure	63.8	67.1	70.6
Binding	60.4	67.3	67.1
Control/Raising	60.9	67.9	66.5
D-N Agr.	72.2	90.8	78.5
Ellipsis	34.4	76.4	62
Filler-Gap.	48.2	63.5	63.8
Irregular Forms	77.6	87.4	67.5
Island Effects	45.6	39.9	48.6
NPI Licensing	47.8	55.9	46.7
Quantifiers	61.2	70.5	59.6
S-V Agr.	65.0	65.4	56.9

Table 1: Performance Metrics of baselines over Different Models

5.2 Smaller Baseline Models

The baselines that BabyLM challange provided were big transformer models with a lot of parameters. This meant that if we were to use them

Parameter	BabyLM Babeline's values
learning rate	$1e-4$
training epochs	400
base batch size	16
desired batch size	128
weight decay	0.1

Table 2: BabyLM Baseline Parameters

metric	Dummy T5
Anaphor Agr.	60.63
Agr. Structure	59.49
Binding	48.01
Control/Raising	50.35
D-N Agr.	50.21
Ellipsis	42.73
Filler-Gap.	50.68
Irregular Forms	54.20
Island Effects	49.51
NPI Licensing	34.85
Quantifiers	67.21
S-V Agr.	50.19

Table 3: Performance Metrics of t5 baseline

as baselines to test our proposed approaches, we would need to work with large models on the scale of their baselines. Due to the computational (no easy access to powerful GPUs) and time constraints we are working under, we have decided to experiment on smaller transformer models which would allow us to conduct our experiments more easily. As a result, we needed baselines which made use of smaller models.

5.2.1 Model Selection

To this end, we have decided to scale down the popular transformer models available in the HuggingFace ecosystem and train them to come up with our baselines. To achieve this, first we decided to experiment with BERT, and GPT2 models. We chose these two architectures due to the following reasons. To begin with, they have an existing easily configurable implementation available in the HuggingFace API. This mean that we can easily scale these models down via their supported configurations, and build our models on top of them since at their core these models are basically an instance of the *torch.nn.Module*. Moreover, BERT is an encoder transformer, and GPT2 is a decoder transformer. We wanted to test the effects of our approach on different transformer architectures with different pretraining objectives to have a more comprehensive understanding of the effects of our methodology. Additionally, these models are among the very inspiring and popular models in the field of natural language processing.

5.2.2 Parameters for Scaling Down

The parameters that our baseline models have settled to have can be found in the table 4. In order to train our baselines we needed a tokenizer. We have trained a tokenizer from scratch on our corpus. This allowed us to learn the structure in the provided BabyLM dataset and have a more domain specific tokenization which will be beneficial to the representaitons that our models will learn. Also, we have reduced the vocabulary size of the tokenizer to 2000 which allowed us to reduce the vocabulary size of our models. This reduced the number of embeddings our models learned, and the size of the output features of the final prediction linear layer of our models (i.e. model assigns probabilities over a smaller vocabulary). In order to eliminate the requirement of having to train our tokenizer on the whole available data, we have decided to gather a representative sample dataset to train our tokenizer on. For this we have randomly picked samples from the available data. We have fine tuned these parameters by investigating the amount of tokens produced per a given sequence. For example we did not want our tokenizer to overfit on the training data and produce a token per every word, nor did we want our tokenizer to underfit and produce too many meaninless tokens per words. By considering this trade-off and the amount of computational time required, we have settled for the length (representative sample size) of 10000, and vocabulary size of 2000. For the tokenizer we have experimented with both BERT's and GPT2's tokenizers, and decided to use GPT2's Byte-Pair-Encoding (BPE) in the end. We tried to reduce the capacity of the model as little as possible while meeting our aforementioned criteria. We reduced the hidden size, number of hidden layers, number of attention heads, and vocabulary size of our model by playing with these parameters and running the training loop to see how long it took for an epoch to finish. After settling on their values, we have run a simple grid search

over a few learning rate candidates in the range $[1e-3, 1e-6]$. We ran both BERT and GPT2 models with these parameters and got best results (stable decrease, and convergence in the training and validation loss curves) for the learning rate values around $5e-5$ hence we chose it as our learning rate for training our baseline models.

Parameter	Scaled Down Value (Our Baseline's)
learning rate	$3e-5$
training epochs	3
batch size	16
number of attention heads	4
number of hidden layers	4
hidden size	256
vocabulary size	2000 + special tokens
maximum block size	128

Table 4: Scaled Down Baseline Parameters

5.2.3 Training Baselines

We have trained our baseline models using all of the available training data (i.e. whole dataset). In our approach we investigate the effects of curriculum learning. Because of this, we believe that having baselines trained on the whole dataset yields proper comparisons for our experiments. The figures below (6, 5, 7, and 2) list the losses that our baseline models (BERT_baseline, GPT2_baseline) got during their training, validation (dev), and testing runs. Note that even though the results indicate that both of our baseline models can be improved if trained longer (validaiton and training losses have not converged and keep decreasing in the results), we take these results as is and do not train longer due to the aforementioned resource restrictions we work under. Additionally, all of the training procedures for these baselines including the optimizers, architecture of the models, and the pretraining objectives used, all follow the explained details in the section 6.

6 Your approach

Our challange required us to pretrain language models in an efficient way given a small dataset. This dataset resembles the language information that a children is exposed to. The challange is

Epoch	Training Loss	Validation Loss
1	4.58	4.05
2	4.01	3.86
3	3.87	3.81

Table 5: Training and Validation Losses During the Training of Our GPT2 Baseline

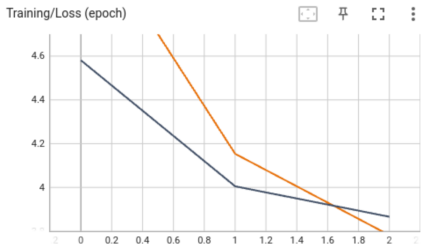
Epoch	Training Loss	Validation Loss
1	5.25	4.21
2	4.15	3.69
3	3.78	3.54

Table 6: Training and Validation Losses During the Training of Our BERT Baseline

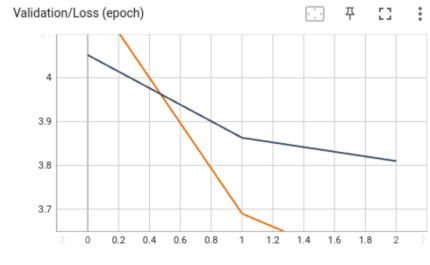
Baseline	Test Loss
GPT2	3.71
BERT	3.44

Table 7: Training and Validation Losses During the Training of Our BERT Baseline

inspired by the fact that small children are exposed to much smaller data than current large language models are trained on, hence we should focus on pretraining on small datasets. Inspired by this observation, we thought since we are opting for learning from data that a child learns from, maybe our models will do better if they learn to gradually accomplish more complex tasks similar to the way a child learns. This lead to us proposing curriculum learning and investigating it from various aspects such as its scalability and sensitivity to choice of transformer architecture it is used with. We expected our proposed approach using curriculum learning would strive and achieve substantially better results than our baseline models. Additionally, we believed that using curriculum learning and training on easier data first would allow the model to make more of the harder dataset in comparison to other approaches due to the scarcity of the available data, and the gradual approach to learning allowing the model to learn language which is a very complex concept to smaller more digestible parts, eventually yielding stable training. We have managed to complete a working implementation that supports the experiments we wanted to conduct. On the other hand,



(a) Training Loss



(b) Validation Loss

Figure 2: Training and Validation Loss Plots (Orange line corresponds to BERT baseline, and grey one correspond to GPT2 baseline)

we did not have access to powerful GPUs which hindered our experimentation process and considerably shaped our experimentation process. The following subsection will go into further detail regarding the subject.

6.0.1 Proposed Approach

We worked on the strict small task of the BabyLM challange which provided a small dataset to train our models on. It prohibited the use of any data other than the ones they have specifically provided. This included using the pretrained models which are already available on the HuggingFace since they were pretrained on large but different datasets than the ones that they have provided. We believe that due to the scarcity of the available data, simply training a language model on the whole corpus would not do very well. To this end, we have decided to exploit the structure in the datasets. We observed that the data provided by the challange has an eminent complexity (language wise advancement level). For example,the data provided in the *aochildes* dataset has short sentences with very basic vocabulary. On the other hand, the *wikipedia* dataset is comprised of long and complex sentences which are from time to time embroidered with advanced vocabulary. Inspired by this, we proposed the use of curriculum learning for training the models on the small dataset provided by the challange. We believe that due to the scarcity of the data, training models on the whole dataset at once is not feasible for the trained model since it is not enough to discover the complexities of the language.

6.0.2 Use of Data for Curriculum Learning

There can be different approaches to curriculum learning as observed in the section 3, but underneath all lies the idea of training models on varying levels of difficulty in consecutive iterations. This leads models to gradually learn to accomplish more complex tasks starting from the more simpler ones since the simple skills are probably required for the harder task too. This approach has strong resemblences to the way that humans learn by which the simpler skills are acquired first to do the harder ones. For example, at schools children are gradually taught harder subjects as they get older and pass their courses. Our appraoch divides the available training dataset to two categories; easier training dataset, and harder training dataset. The way we create these two splits is based on the qualitative observations we made on the data files as indicated below.

Easier Dataset Files:

- *aochildes.train*:
  1. Short sentences. *Example: "come here. say mama"*
  2. Repetition of simple sounds and words. *Example: "bang bang. "*
  3. Basic vocabulary *Example: "hello. hi?"*
- *open_subtitles.train*:
  1. Short and simple structured sentences. *Example: "what happened? This is Vinnie."*
  2. Basic vocabulary *Example: "Hello? It's Sandy"*
- *qed.train*:
  1. Short and simple structured sentences. *Example: "GRAHAM! COME ON, GRAHAM! GRAHAM! (cries) SO HAPPY YOU COULD COME. SON. "*
- *simple_wikipedia.train*:
  1. There is also a harder dataset gathered from wikipedia, *wikipedia.train* which we placed in the harder dataset split.

- *switchboard.train*:

    1. Simple sentence structure (harder than the previous ones but still simpler thant the ones we picked for the harder dataset). *Example: "I'm ready for it, but they're predicting some more snow for our direction."*

Harder Dataset Files:

- *bnc_spoken.train*:

    1. Very long and complex sentences. *Example: "What about the skills which are very important but don't come as a formal part of any subject, the so-called communication skills?"*

- *cbt.train*:

    1. Advanced vocabulary, very long and complex sentences. *Example: "One was another reminded the queen of somebody she did not like ; but at length an idea flashed into the queen 's head , and she called out : ' I know !"*

- *children_stories.train*:

    1. Complex sentence structure and vocabulary. *Example: "As he was thus thinking he let the bridle fall, and the horse began to caper about, so that he was continually disturbed in his meditations, and could not collect his thoughts at all."*

- *gutenberg.train*:

    1. Harder vocabulary with long sentences. *Example: "business puzzles me, though you and your friends here seem to find it devilish amusing."*

- *wikipedia.train*:

    1. There is also an easier dataset gathered from wikipedia, *simple_wikipedia.train* which we placed in the easier dataset split.

6.0.3 Curriculum Learning

Our implementation of code is specifically designed to easily support curriculum learning by means of gradually training the model on different datasets of your liking. By leveraging this functionality, we experimented with curriculum learning the following ways. First curriculum learning approach takes the stance of training the model first on the easier dataset and then the harder one. In order to test this, we have tried first training our models using the *easier dataset*, and continue training the same model on the *harder dataset*. These are the datasets that we have defined earlier in 6.0.2. Additionally, we have tested the impact of doing the opposite of the first approach to further understand how it impacts the model training dynamics. In order to do this, we have also trained models starting with the *harder dataset*, then moving on to training on *easier dataset*.

6.0.4 Model Architectures

We have tested our approach using two different transformer architectures, namely; encoder transformer, and decoder transformer architectures. For the encoder family we have picked the Bidirectional Encoder Representations from Transformer (BERT) model. We have picked this model due to its popularity, relatively smaller number of parametesrs, and the huge impact it once had on the natural language processing domain. Our custom encoder model, loads the body of the BERT model from the HuggingFace transformers module (randomly initialized) and places a linear layer with the output dimension of vocabulary size to act as the classification head over the vocabulary. We pretrain this model using the masked language modeling (mlm) objective using the data specially preprocessed for this task as was explained in the 4.1. For the decoder family, we have picked the Generative Pretrained Transformer 2 (GPT2) model. We have picked this model due to the recent success of ChatGPT which is makes use of a GPT based model like the GPT2 model. Even though GPT2 is far less capable of such large language models today, we deemed it enough for the small language modeling task we have at our hand. Our custom decoder model loads the body of the GPT2 model frmo the HuggingFAce transformers module (randomly initialized) and places a linear layer with the outut dimension of vocabulary size to act as the classificaiton head over the vocabulary similar to what we did for the encoder based model. We pretrain this model using the causal language modeling (clm) objective using the data specially prepared for this task as was explained in the 4.1.

6.0.5 Different Scale Models

In order to investigate the effects of curriculum learning thoroughly, we have experimented with models at different scales (e.g. number of model parameters, depth, complexity). We have introduced two model scales; small models, and large models. It is important to note that the prefix *large* in the *large models* hints on the fact that these models are relatively larger than the smaller ones. We do not want to give the wrong impression that these models are large in the sense that these models can be considered large in the todays standards. We have worked on way smaller model scales in our project due to restrictions that we had to work under such as the lack of computational resources to work with models that are large in todays standards. Doing so would have impeded with our budget, drastically introduced the number of experiments we can conduct, and impeded with our ability of recovering from bugs introduced during our code development phase. The parameters that we used for the small models, and the large models are listed in the table 8. During scaling up we wanted to increase the complexity of the model hence increasing its ability to learn (represent). In order to achieve this, we have increased the number of attention heads from. We believe that increasing it yielded the model the capacity to learn different relations between the tokens in the input. This was motivated by the idea that the different heads in a given multihead attention mechanism learns to represent different relations between the inputs which are useful for the model to succeed in its task. Furthermore, we have increased the depth of the model. In the deep learning applications, usually increasing the depth of the model improves the models performance more than increasing the models width. Because of this, we decided to increase the number of hidden layers and not the hidden size of the model. Due to computational scarcity we faced, we could not afford to increase both the hidden size parameter and and the number of hidden layers. In light of the argument we previously explained, we chose to only increase the depth of our model. In order to match the increased expressivity of our model during its training phase, we have doubled the number of epochs and increased the learning rate.

Parameter	Small Models	Large Models
learning rate	$3e-5$	$1e-4$
training epochs	3	6
batch size	16	32
number of attention heads	4	8
number of hidden layers	4	5
hidden size	256	256
vocabulary size	2000 + special tokens	2000 + special tokens
maximum block size	128	128

Table 8: Hyperparameters for the Small and Large Scale Models We Used

6.0.6 Experiment Design

To motivate the way we constructed our experiments, let us first go through some of the variable design choices we have control over.

- *Model Architecture*: We can choose from different transformer architectures which make use of different training objectives. Supporting different architectures will allow us to investigate the effects of our curriculum approach on different family of transformers.
  - Encoder Model: To this end, we have decided to use the body of the BERT model with masked language modeling pretraining objective as the model from the encoder family of transformers.
  - Decoder Model: To this end, we have decided to use the body of the GPT2 model with causal language modeling pretraining objective as the model from the decoder family of transformers.
- *Scale of the Model*: Experimenting with the same models on different scales could give use an idea of how our approach scales with the complexity of the model.
  - Small Models: These models are relatively smaller than the large models we have (e.g. our Small GPT based, and small BERT based models).
  - Large Models: These models are relatively larger than the small models we have (e.g. our Large GPT based, and large BERT based models).

- *Curriculum Learning Approach*: Various methodologies can be develop that fall under the idea of curriculum learning. We have designed our curriculum learning experiments based on the difficulty of the dataset the model is trained on in two separate training stages.
  - Easy First, Harder Second Approach: In this approach, the randomly initialized model will initially be trained on the easier dataset. Then, the trained model will be loaded and trained using the harder dataset next. One can refer to 6.0.2 for a detailed explanation of how these datasets were constructed.
  - Hard First, Easy Second Approach: In this approach, the randomly initialized model will initially be trained on the harder dataset. Then, the trained model will be loaded and trained using the harder easier dataset next. One can refer to 6.0.2 for a detailed explanation of how these datasets were constructed.

Below we go over the steps we followed for our experiments which we have specifically designed following the short discussion we have presented earlier:

1. ***Step 1: Tokenizer***: In this step, we train a Byte-Pair-Encoding tokenizer from scratch on a representative sample dataset we have gathered from the available training dataset. The reasoning behind the parameters that we settled for is discussed in 5.2.2. In short, at this stage we train a tokenizer which has a smaller vocabulary size in comparison to the pretrained tokenizers that come with the models we used from the HuggingFace ecosystem ($\sim$500000 vocabulary size for *bert-base-uncased*, and *GPT2*).

2. ***Step 2: Small GPT2***: We initialize a small GPT2 model and pretrain it on the whole training dataset using the causal language modeling objective. Note that you can refer to table 8 for the training and model parameters used for the Small and Large models mentioned. Also, during all of the steps, the general pipeline we follow is as follows. During training, after each epoch we evaluate the performance of our model on the validation dataset. After the training is completed, we evaluate the models performance on the test dataset.
   - ***Step 2_1***: Training of the small GPT2 model on the whole training dataset (includes validation on the validation dataset too).
   - ***Step 2_2***: Testing of the trained small GPT2 model on the testing dataset.

3. ***Step 3: Small GPT2 + Curriculum Learning***: We initialize a small GPT2 model and pretrain it using the curriculum approach we have motivated earlier. We first train the initialized model on the easier dataset, then load the pretrained model and keep training it on the harder dataset. Note that the datasets we have mentioned here follow from 6.0.2.
   - ***Step 3_1***: Initialization and training of the small GPT2 model on the easier dataset (includes validation on the validation dataset too).
   - ***Step 3_2***: Loading the model trained during the *Step 3_1*. The loaded model is continued to be trained on the harder dataset. After this step the model has made use of the whole training dataset that is available using curriculum learning.
   - ***Step 3_3***: Testing the small GPT2 model trained with curriculum learning.

4. ***Step 4: Small BERT***: We initialize a small BERT model and pretrain it on the whole training dataset using the masked language modeling objective.
   - ***Step 4_1***: Training of the small BERT model on the whole training dataset (includes validation on the validation dataset too).
   - ***Step 4_2***: Testing of the trained small BERT model on the testing dataset.

5. ***Step 5: Small BERT + Curriculum Learning***: We initialize a small BERT model and pretrain it using the curriculum approach we have motivated earlier. We first train the initialized model on the easier dataset, then load the pretrained model and keep training it on the harder dataset.
   - ***Step 5_1***: Initialization and training of the small BERT model on the easier

dataset (includes validation on the validation dataset too).

- ***Step 5_2***: Loading the model trained during the *Step 3_1*. The loaded model is continued to be trained on the harder dataset. After this step the model has made use of the whole training dataset that is available using curriculum learning.
- ***Step 5_3***: Testing the small BERT model trained with curriculum learning.

6. ***Step 6: Large GPT2 (Scaled Up)***: We initialize a large (scaled up) GPT2 model and pretrain it on the whole training dataset using the causal language modeling objective. This model will act as the baseline model for the larger models we train while investigating the effect that curriculum has on the scaled up models. This will help us gain intuition on the dynamics of how our proposed approach scales with the model size.

    - ***Step 6_1***: Training of the large GPT2 model on the whole training dataset (includes validation on the validation dataset too).
    - ***Step 6_2***: Testing of the trained large GPT2 model on the testing dataset.

7. ***Step 7: Large GPT2 + Curriculum Learning (Approach: easy first then hard)***: We initialize a large GPT2 model and pretrain it using the curriculum approach we have motivated earlier. We first train the initialized model on the easier dataset, then load the pretrained model and keep training it on the harder dataset.

    - ***Step 7_1***: Initialization and training of the large GPT2 model on the easier dataset (includes validation on the validation dataset too).
    - ***Step 7_2***: Loading the model trained during the *Step 7_1*. The loaded model is continued to be trained on the harder dataset. After this step the model has made use of the whole training dataset that is available using curriculum learning.
    - ***Step 7_3***: Testing the large GPT2 model trained with curriculum learning.

8. ***Step 8: Large GPT2 + Curriculum Learning (Approach: hard first then easy)***: We initialize a large GPT2 model and pretrain it using the curriculum approach we have motivated earlier. We first train the initialized model on the harder dataset, then load the pretrained model and keep training it on the easier dataset.

    - ***Step 8_1***: Initialization and training of the large GPT2 model on the harder dataset (includes validation on the validation dataset too).
    - ***Step 8_2***: Loading the model trained during the *Step 8_1*. The loaded model is continued to be trained on the easier dataset. After this step the model has made use of the whole training dataset that is available using curriculum learning.
    - ***Step 8_3***: Testing the large GPT2 model trained with curriculum learning.

6.0.7 Experimental Results

In the table 9 and the figures 3, 4, 5, 6 below, we share the results that our model achieved on the training, validation, and test datasets. Then we proceed to explain our hypothesis and the conclusions we derived under the light of our findings. Note that the BabyLM Challange also provides an evaluation pipeline which can be run with our trained models to perform further investigation. We leave the analysis of these results to section 7.

- ***Impact of Curriculum Learning on Small Scale Model Training***: Our hypothesis was that using curriculum learning with small scale models while training on the strict small dataset of BabyLM Challange would yield better performance. By comparing results from *Step 2* and *Step 3*, we observe that both the training and the validation curves are decreasing and they have not converged yet. This hints us that both decoder family models have underfit the training dataset and could have benefitted from being trained longer. But due to the aforementioned constraints we take these results as is and comment on their results without further experimentation on the case (Note that this applies for all of the remaining sections and the arguments

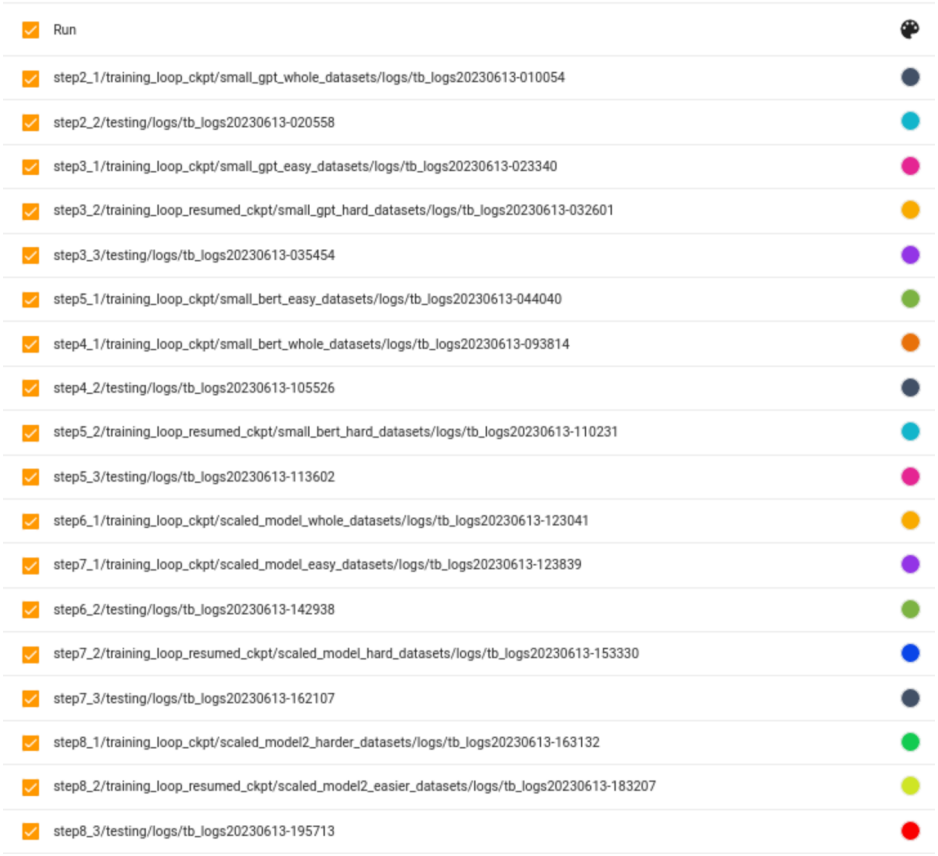


Figure 3: Legend Used for the Training, Validation, and Test Curves

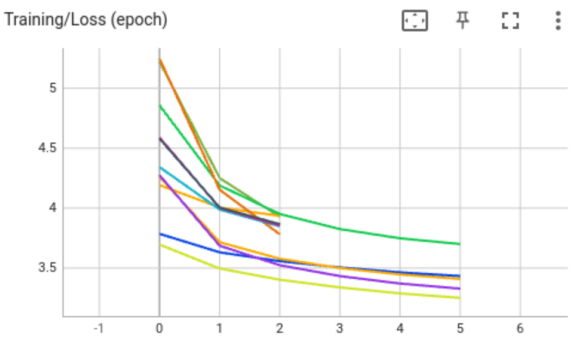


Figure 4: Training Loss Curve for the Experiments

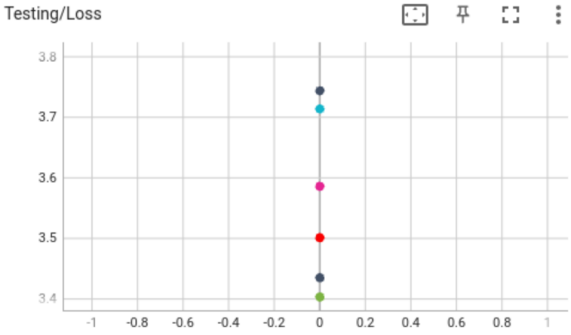


Figure 6: Test Losses for the Experiments

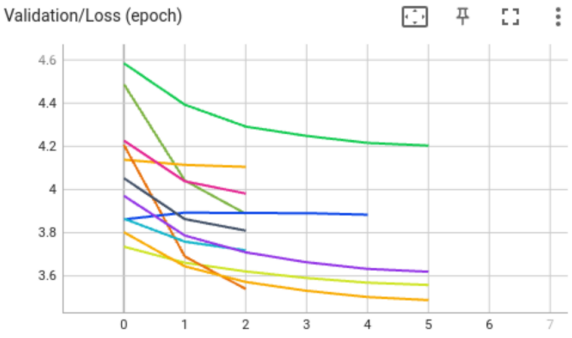


Figure 5: Validation Loss Curve for the Experiments

made in them too). From their corresponding results on the test dataset, we see that small GPT2 model with curriculum learning has done worse than its counter part which was trained on the whole dataset. From this observation, we infer that our claim fails for the small scale decoder based models. Additionally, we conducted a similar experiment using BERT based models which belong to the encoder family of transformers (different than the previous decoder family (e.g. GPT2)). For this we conducted the experiments *Step*

Step	Training Losses	Validation Losses	Test Losses
2_1, 2_2, 2_3	{4.58, 4.01, 3.87}	{4.05, 3.86, 3.81}	3.71
3_1	{5.49, 4.0, 3.86}	{4.23, 4.04, 3.98}	-
3_2	{4.19, 4.01, 3.94}	{4.14, 4.11, 4.10}	-
3_3	-	-	3.97
4_1, 4_2, 4_3	{5.25, 4.15, 3.78}	{4.21, 3.69, 3.54}	3.44
5_1	{5.22, 4.25, 3.92}	{4.48, 4.04, 3.88}	-
5_2	{4.34, 3.99, 3.85}	{3.87, 3.76, 3.72}	-
5_3	-	-	3.58
6_1, 6_2, 6_3	{4.26, 3.72, 3.58, 3.53, 3.45, 3.41}	{3.80, 3.64, 3.57, 3.53, 3.50, 3.49}	3.40
7_1	{4.23, 3.69, 3.52, 3.43, 3.37, 3.33}	{3.97, 3.78, 3.70, 3.66, 3.63, 3.62}	-
7_2	{3.79, 3.63, 3.56, 3.51, 3.46, 3.43}	{3.86, 3.89, 3.89, 3.88, 3.89}	-
7_3	-	-	3.74
8_1	{4.86, 4.19, 3.95, 3.83, 3.75, 3.70}	{4.58, 4.39, 4.29, 4.25, 4.22, 4.20}	-
8_2	{3.70, 3.50, 3.40, 3.34, 3.29, 3.25}	{3.74, 3.66, 3.59, 3.57, 3.56}	-
8_3	-	-	3.50

Table 9: Training, Validation, and Test Results for the Experiments

4 and *Step 5*. By comparing the performance they had on the held out test dataset, we observe that the encoder based model trained with curriculum learning has done worse than its counterpart trained on the whole dataset. By interpolating the results we got by experimenting with two models which belong to two different family of transformers, our hypothesis was wrong. The results we got indicate that training small scale transformer models on the given dataset does not benefit from our proposed curriculum learning approach. On the other hand, we see that both training and validation curves were decreasing for all of the aforementioned models. We suspect that if our experiments were to be conducted with increased training epochs (i.e. if trained for longer), curriculum learning approach can yield better results.

- ***Impact of Curriculum Learning on Large Scale Model Training***: Our hypothesis was that using curriculum learning with large scale (relative to our experiments) models while training on the strict small dataset of BabyLM Challange would yield better performance. By comparing results from *Step 6* and *Step 7*, we observe that both the training and the validation curves are decreasing and they have not converged yet. From their corresponding results on the test dataset, we see that large GPT2 model with curriculum learning has done worse than its counter part which was trained on the whole dataset. From this observation, and by interpolating our previous result that using different family of transformer models does not change the outcome of our proposed approach, we come to the conclusion that our claim fails for the large scale models. In other words, our hypothesis did not hold. The results we got indicate that training large scale transformer models on the given dataset does not benefit from our proposed curriculum learning approach. On the other hand, we again see that both training and validation curves were decreasing for all of the aforementioned models. We believe that if our experiments were to be conducted with increased training epochs (i.e. if trained for longer), curriculum learning approach has the potential to yield better results eventually.

- ***Impact of Different Curriculum Learning Approaches***: We wanted to explore the effects of different approaches to curriculum learning. For our experiments we picked two approaches. The first one (approach 1) is training the model on the easier dataset first, then continuing training on the harder dataset. The second approach (approach 2) is training the model on the harder dataset first, then continuing training on the easier

dataset. For this purpose, we have designed and conducted *Step 7*, and *Step 8*. The idea was to train two models which are initialized and trained under the same conditions except the fact that they utilized different approaches to curriculum learning. The validation results from *Step 7* (approach 1) indicate that while training the model on the harder dataset, the the valdation losses did not improve and got first during the first epoch. This contradicts what we expected from curriculum learning. We believed that the simpler relations learned during the training on the easier dataset would help model perform considerably better during its continued training on the harder dataset. On the contrary, we see that its performance on the test dataset has deteriorated from 3.62 to 3.89. It is important to note that for both stages of the curriculum learning training we validate our results using all of the available validation dataset and not just the validation dataset that corresponds to the training dataset that the model is being trained at a given step. This means that this unexpected contradiction with our hypothesis cannot be simply alluded to the difference in the validation dataset. In addition to this, we observe a similar pattern in the results of the *Step 8*, where validation loss deteriorates between two phases of training again (from 4.22 to 3.56). For both of the aforementioned cases, we see that traing losses were decreasing regardless of the discrepancies in the validation losses. Because of this, we believe that in *Step 7* and *Step 8* our models have suffered from overfitting the dataset which explains the decreasing loss on the dataset which is not accompanied by the validation loss. We suspect that the increased complexity of the model allowed it to overfit on the training dataset. We suggest repeating these steps with decreasing the learning rate, increasing batch size, reducing the complexity of the model (e.g. reduce hidden dimension, number of layers, number of attention heads). On the other hand, putting these observations aside, the test losses of these models indicated that training first on easier dataset, then training on harder dataset (approach 1) performs worse than the opposite approach (approach 2). Under the light of these results, we suggest the replication our experiments with the adaptation of the second appraoch instead of the first one as further directions to our work.

6.0.8 Implementation Details

Our code implementation follows a modular approach which was built with the ease of experimentation and scalability in mind. It fully supports our needs for the proposed approach and even can be used for different unrelated experiments due to its generalization capabilities. We have extensively made use of *PyTorch*, *HuggingFace models*, and *HuggingFace datasets* for the initialization, training, and data preprocessing stages. Our implementation is comprised of the following files.

- *argument_parser.py*: Adds command line argument parsing capability to our implementation. Using the arguments defined in this file, we can perform our experiments easily. It allows one to specify tokenizers to use with models, or train tokenizers from scratch. Save and load tokenizers and models from both HuggingFace and from local directories. Configure most of the hyperparameters of the models, and specify which datasets to use for training, validating, and testing the models. It is combined with *Makefile* to ease the experimentation process and contribute to the reproduciblity of our work. An example for its use can be found in 7.

- *custom_models.py*: It implements the *CustomPretrainingTransformerModel* class which we used to create, save, and load our custom transformer models. It extends *PreTrainedModel* class of HuggingFace which extends *torch.nn.Module* class underneath. It can load transformer models from HuggingFace and use their body as its own encoder. It also initializes a linear layer which is to be used as the classification head over the target vocabulary.

- *huggingface_trainer_loop.py*: It adds support for training the models using the HuggingFace's *Trainer*. Our implementation supports both training using huggingface *Trainer*, and using our custom training loop written in *PyTorch*. These two options can be used interchangebly by simply passing the –*torch_training* parameter supported by the *argument_parser.py*. In all of our experiments

we have used our custom *PyTorch* training loop.

- *Makefile*: Makefile is not an integral part of our code implementation. In other words removal of it would not cause any harm. We have used it for the sole purpose of having an easier way of configuring the command line arguments that are supported by *argument_parser.py*. We have taken it a step further by extending with all of the python calls one needs to reproduce our experiments. It has commands that correspond to the names of the steps of our experiments (e.g. *Step 1*, *Step 7_1*). Refer to 7 for an example.

- *pretraining_dataset.py*: It loads specified datasets and creates training, validation, and testing datasets. Given a tokenizer and a dataset, it tokenizes the datasets. In addition to these it is responsible for the implementation of the data collators for the causal language modeling objective and the masked language modeling objective. It most notably relies on the HuggingFace's *datasets* module and its supported *map* functionality.

- *pytorch_training_loop.py*: It is a replacement for the *huggingface_trainer_loop.py* functionality. We found HuggingFace's *Trainer* implementation to be hiding too much from us and hard to extend, thus we decided to implement a training loop ourselves using *PyTorch*. It is used for training a model, evaluating it after every epoch. It keeps track of the model with the best validation performance and saves it as checkpoint. This acts in a similar manner to early stopping. It defines an *torch.optim.AdamW* optimizer, and implements gradient normalization to train the model with the hopes of avoiding exploding gradients problem, and calls an instance of our custom *Logger* class to log the appropriate information on the supported mediums. It also implements the functionality for testing a model on a given test dataset.

- *tokenizer.py*: It adds the functionality to load, save, train tokenizers from both HuggingFace and local paths.

- *utils.py*: Motivated by the fact that training transformer models have high time and compute requirements which we were struggling with, we decided that logging was of uppermost importance to us. To this end, we have developed this module which implements a custom *Logger* class. This class extends python's *logging* module and makes use of *TensorBoard*. It supports logging to command line, files, and *TensorBoard logs*. We made extensive use of it in order to have a better understanding of the learning dynamics that are taking place in our experiments, to finding bugs in our code during the earlier stages of its development phase, and for keeping track of the hyperparameters we have used during our experiments. Refer to 8 for an example log written to a file, and 4 for an example log written as a *TensorBoard log*. In our experiments it creates a directory called *save_dir*, and logs the experiments under their corresponding name. For instance, running *Step2_1* would create the directory *./save_dir/step_1*. It is this directory in which we have also saved our model checkpoints during the training.

7 Error analysis

10 shows the evaluation pipeline results of our small scaled models. The comparison of our models with the large scale baselines show that our models with fewer transformer layers with considerably smaller vocabulary size performs poorer in classification tasks in comparison to the baselines. We explain this as follows. Our models were trained for a significantly smaller number of epochs when compared to the baselines. This limited training time may have resulted in incomplete convergence (underfitting) and affected the models' ability to capture intricate patterns and features which are latent in the data. Additionally, the large-scale baselines utilize the full capacity of large language models, enabling them to capture a broader range of linguistic nuances and dependencies. In contrast, our models have a limited capacity due to the reduced number of transformer layers and a smaller vocabulary size. This limitation could result in less expressive power, leading to poorer performance in complex classification tasks. However, despite these limitations, our models have demonstrated comparable results to the baselines in certain tasks, such as Ellipsis and Binding. When we compare our small scaled

```
step5_1: # Train using curriculum learning approach 1(train on easier data first)
    python3 argument_parser.py --create_model_load_tokenizer_train_and_save_model True \
    --torch_training True \
    --train_dataset_file_names ${easier_training_dataset_file_names} \
    --validation_dataset_file_names ${all_dev_dataset_file_names} \
    --test_dataset_file_names ${all_test_dataset_file_names} \
    --transformer_model_name bert-base-uncased --pretraining_task mlm \
    --training_batch_size 16 --num_workers 0 \
    -lr "3e-5" --grad_norm_clip 1.0 --num_epochs 3 \
    --hidden_size 256 --num_attention_heads 4 --num_hidden_layers 4 \
    --model_checkpoint_path "./save_dir/step5_1/training_loop_ckpt/small_bert_easy_datasets"

step5_2: # Train using curriculum learning approach 1 cont.(train on harder data now)
    python3 argument_parser.py --load_model_load_tokenizer_and_train True \
    --torch_training True \
    --train_dataset_file_names ${harder_training_dataset_file_names} \
    --validation_dataset_file_names ${all_dev_dataset_file_names} \
    --test_dataset_file_names ${all_test_dataset_file_names} \
    --tokenizer_save_or_load_path "./save_dir/saved_tokenizer" \
    --model_load_path "./save_dir/step5_1/training_loop_ckpt/small_bert_easy_datasets/2" \
    --model_checkpoint_path "./save_dir/step5_2/training_loop_resumed_ckpt/small_bert_hard_datasets" \
    --pretraining_task mlm \
    --training_batch_size 16 --num_workers 0 \
    -lr "3e-5" --grad_norm_clip 1.0 --num_epochs 3

step5_3: # Test the small BERT model with curriculum learning
    python3 argument_parser.py --load_model_load_tokenizer_and_test  True \
    --tokenizer_save_or_load_path "./save_dir/saved_tokenizer" \
    --model_load_path "./save_dir/step5_2/training_loop_resumed_ckpt/small_bert_hard_datasets/2" --pretraining_task mlm \
    --model_checkpoint_path "./save_dir/step5_3/testing" \
    --testing_batch_size 16 --num_workers 0
```

Figure 7: Example Taken from *Makefile* Which Can be Called to Reproduce *Step5_1*, *Step5_2*, *Step5_3*

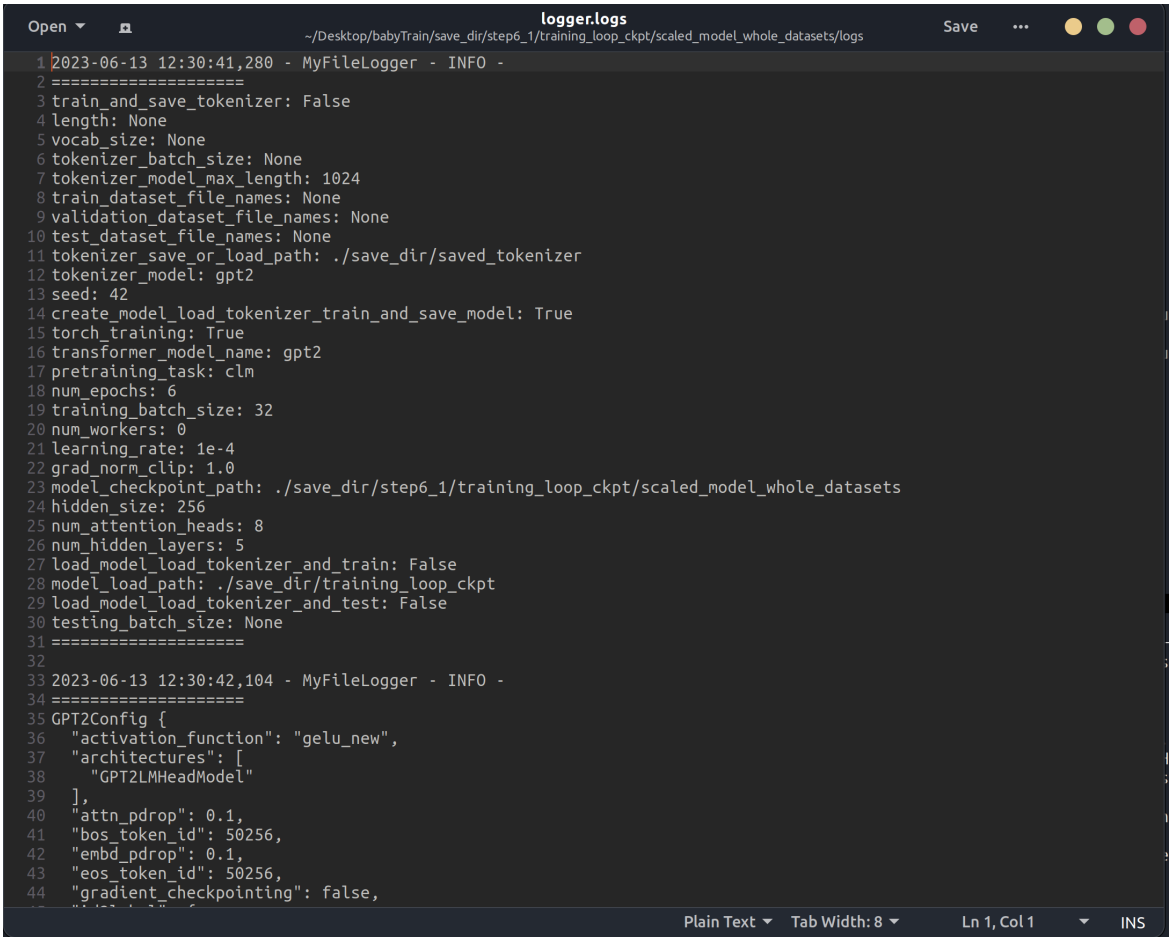


Figure 8: Example That Demonstrates a Section From a Sample Log File Written During the *Step6_1*

models with the baselines, our models are not good at Irregular Forms, NPI Licensing and Quantifiers. Irregular forms have been included in the dataset. It contains dialogues of the child where informal sentence or word phrases have been utilized by the child, however in language modeling task the model can't learn this task easily since the words in childhood are an abrupt reaction of the child. Since our models were trained with casual language modeling and masked language modeling objectives, they reach reasonable performance in subject verb agreement and filler gap tasks. This suggests that our models are capable of capturing specific linguistic phenomena effectively, even under their constrained condition. We believe that if these models can be trained for longer epochs using our proposed curriculum learning based approach, they can learn to capture the patterns in the dataset, and show better results than the baselines. Under such conditions, its results will be consistent with the works of the others, such as the ones that were mentioned in 1.

metric	Step2	Step 3	Step 4
Anaphor Agr.	51.38	42.33	48.31
Agr. Structure	52.90	52.80	51.41
Binding	64.54	57.81	62.21
Control/Raising	53.03	53.23	53.65
D-N Agr.	50.57	50.45	49.43
Ellipsis	40.36	40.36	39.43
Filler-Gap.	50.03	44.99	46.90
Irregular Forms	53.18	50.13	43.87
Island Effects	51.46	48.21	46.82
NPI Licensing	30.96	35.27	37.02
Quantifiers	30.78	44.62	39.05
S-V Agr.	50.42	51.10	48.83

Table 10: Performance small sized corpus steps

8 Contributions of group members

- Can Gözpınar: Implemented the code base from scratch. Designed the experiments. Conducted the training, validation, and testing of the models for the experiments.
- Aydın Ahmadi: Responsible of the evaluation and interpretation of the trained models using the evaluation pipeline provided by the BabyLM Challange.

9 Conclusion

We have proposed and investigated the adoption of a curriculum learning based approach for the BabyLM challange. Unlike our hypothesis, our proposed approach did not yield superior results nor it beat baselines. We mostly allude these results to the fact that we trained for considerably small epochs due to our computational and time constraints. We believe that this resulted in underfitting and hindered our models ability to leverage the benefits of curriculum learning. As further direction, we suggest the repetition of our experiments with larger models which are trained for longer number of epochs. Under these circumstances we think that curriculum learning would provide a considerably stable training experience which would help it surpass the other methods on our small language modeling task. Furthermore, one can experiment with different approaches to curriculum learning than the two approaches we have explored.

10 AI Disclosure

Aydın has used AI assistance to rephrase his sentences and paragraphs. The prompts were like rephrase this paragraph in a better and coherent manner. LLM's sometimes just make the paragraphs and sentences much longer while adding words and phrases that are not redundant.

Limitations

We encountered several limitations during our project that are worth discussing in this section. One notable limitation is the utilization of large-scale transformers in our models, which incorporate deep hidden layers. These architectures, while beneficial for capturing complex patterns, posed challenges in terms of training time and scalability. Due to time constraints, we were unable to extend our approaches to larger domains and fully leverage the potential of the models' architecture for extensive experimentation.
Furthermore, our models were trained for fewer epochs compared to the baselines, and their training time cannot be directly compared. Notably, even the finetuning task introduced by the BabyLM community involved training for more epochs than our models. However, it is important to note that our models exhibited converging loss, indicating that they were learning over

time. Therefore, we posit that training our models for a longer duration and on a larger portion of the dataset would likely improve their performance and scalability across diverse tasks within a broader domain. We also wanted to experiment different strategies of curriculum learning which have been explored in the literature, such as training the models with hardness scoring function.
It is worth acknowledging that the limitations we faced hindered our ability to achieve results in a shorter timeframe and restricted our exploration of more extensive experiments. Despite these limitations, we believe that with extended training and exposure to a larger dataset, our models would demonstrate scalability to larger domains and yield favorable results across various tasks.

Ethics Statement

Our models contain biased results since it has been trained on a small-scale of the dataset and can not be generalized on the lines which has not been trained on. The confidence interval of our results are lower than the confidence interval of trained model on the whole dataset. However, our large scaled models, if can be trained on more epochs with larger domain of the dataset will lie in a lower confidence interval. Our work can be extended with utilizing more computation power to obtain better results.

References

Araujo, V., Villa, A., Mendoza, M., Moens, M.-F., and Soto, A. (2021). Augmenting BERT-style models with predictive coding to improve discourse-level representations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3016–3022, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Hacohen, G. and Weinshall, D. (2019). On the power of curriculum learning in training deep networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2535–2544. PMLR.

Huebner, P. A., Sulem, E., Cynthia, F., and Roth, D. (2021). BabyBERTa: Learning more grammar with small-scale child-directed language. In *Proceedings of the 25th Conference on Computational Natural Language Learning*, pages 624–646, Online. Association for Computational Linguistics.

Ji, H. and Huang, M. (2021). DiscoDVT: Generating long text with discourse-aware discrete variational transformer. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4208–4224, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Lee, H., Hudson, D. A., Lee, K., and Manning, C. D. (2020). Slm: Learning a discourse language representation with sentence unshuffling. In *Conference on Empirical Methods in Natural Language Processing*.

Lee, M., Park, J.-H., Kim, J., Kim, K.-M., and Lee, S. (2022). Efficient pre-training of masked language model via concept-based curriculum masking. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7417–7427, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

van Berkum, J. J. A., Brown, C. M., Zwitserlood, P., Kooijman, V., and Hagoort, P. (2005). Anticipating upcoming words in discourse: evidence from erps and reading times. *Journal of experimental psychology. Learning, memory, and cognition*, 31 3:443–67.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Warstadt, A., Choshen, L., Mueller, A., Williams, A., Wilcox, E., and Zhuang, C. (2023). Call for papers – the babylm challenge: Sample-efficient pretraining on a developmentally plausible corpus.