Accelerating Large Language Model Pretraining via LFR Pedagogy: Learn, Focus, and Review

Anonymous ACL submission

A Appendix and Supplementary Material

A.1 Experiment Details

Datasets The datasets used for our experiments are detailed below:

1. ARC-Challenge (arc, a): A subset of the AI2 Reasoning Challenge with 2,590 challenging multiple-choice science questions designed to test advanced reasoning and knowledge.

2. ARC-Easy (arc, b): A subset of the AI2 Reasoning Challenge with 5,117 relatively easier multiple-choice science questions focusing on basic reasoning and recall.

3. BoolQ (boo, b): A dataset of 16,000+ boolean (yes/no) questions paired with passages, requiring models to infer answers from supporting evidence.

4. HellaSwag (hel): A dataset with 70,000+ multiple-choice questions focused on commonsense reasoning and contextual understanding, particularly in describing scenarios.

5. OpenBookQA (Ope): A multiple-choice question-answering dataset with 5,957 questions requiring knowledge retrieval from a science "open book" and commonsense reasoning.

6. PIQA (Piq): A physical commonsense reasoning dataset with 20,000+ binary-choice questions about everyday situations and physical interactions.

7. Winogrande (win): A dataset with 44,000+ sentence pairs designed to test commonsense reasoning through pronoun disambiguation challenges.

8. WikiText (wik): the WikiText language modeling dataset consists of 100M tokens extracted from Wikipedia articles with high rating. It features two different variants, namely, WikiText-2 and WikiText-103 which differ in the number of tokens and vocabulary size. WikiText-2 consists of 2M tokens and a vocabulary size of 33k whereas WikiText-103 is larger with 103M tokens and a vocabulary size of 267k.

9. LAMBADA (Paperno et al., 2016): the LAMBADA dataset is extracted from the BookCorpus dataset (boo, a) and contains 10k passages. This dataset is useful for testing the ability of an LLM to capture long-range dependencies in text. The objective of this model is to predict the final word in a set of sentences, where humans need at least 50 tokens of context to accurately anticipate the word.

10. One Billion Word Benchmark (Chelba et al., 2014) (1BW): this dataset contains one billion words extracted from the WMT 2011 News Crawl data and is used to measure progress in statistical language modeling.

11. WMT-14 French-English Translation (Artetxe et al., 2018): This dataset contains 36 million training sentence pairs for english to french translation. The test set, which is used for evaluation purposes, consists of 3,003 sentence pairs.

12. Natural Questions (Kwiatkowski et al., 2019): This dataset contains question-answer pairs from Google Search and Wikipedia-based annotations. The training, validation, and test sets consist of 307,372, 7,830, and 7,842 examples.

Models: Tables 1 and 2 describes the different model configurations and pretraining hyperparameters used in LFR for the Llama models.

300M	500M	1.1B	
Layers	12	11	22
#Heads	16	32	32
n_embd	1024	2048	2048

Table 1: Number of layers, attention heads, and the embedding dimensions in the Llama models used for pretraining.

Parameter	Value
Context Length	1024
Embedding Dimension	(768, 1024, 2048)
Total Iterations	100,000
Effective Batch Size	768
Block Size	4096
Weight Decay	1.00E-1
Adam β_1	0.90
Adam β_2	0.95
Warmup Iterations	8000
Minimum Learning Rate	4.00E-5
Maximum Learning Rate	4.00E-04
Learning Rate Schedule	Cosine Decay
Learning Rate Decay Iterations	100,000
GPUs	(4x AMD MI210, 4x AMD MI210, 8x AMD MI250)

Table 2: Pretraining hyperparameters for the Llama 300M-1.1B parameter models. Parameters with multiple values (e.g. Embedding dimensions, batch size, gradient accumulation steps, and GPUs) specified in brackets are for the 300M, 500M, and 1.1B parameter models respectively.

Tables 3 and 4 describes the different model configurations and pretraining hyperparameters used in LFR for the GPT-2 models.

Pretraining: Table 4 shows the hyperparameters for pretraining the GPT-2 124M-1.5B parameter models.

Note that OpenAI pretrained the GPT-2 models using a batch size of 512. Due to insufficient GPU memory, we adjust the number of gradient accumulation steps to achieve the same effective batch size of 512.

	124M	355M	774M	1.5B
Layers	12	24	36	48
#Heads	12	16	20	25
n_embd	768	1024	1280	1600

Table 3: Number of layers, attention heads, and the embedding dimensions in the GPT-2 models used for pretraining.

Finetuning: We use all the same hyperparameters as pretraining, except for the following:

1. Learning rate: 3.00E-5
2. Learning rate schedule: Constant
3. Total iterations: 50

A.2 Limitations and Ethical Considerations

LFR presents the following directions for future work:

1. LFR is evaluated on models up to 1.5B parameters using web-scale datasets such as SlimPajama, constrained by our compute resources. With the clear success on models of such scale, we hope to inspire researchers to validate such focused learning approaches for different model families, and domains.

2. The sensitivity study in this Appendix reveals that the hyperparameters selected in the evaluation section have a large impact on the performance of the trained model. Due to our limited compute budget, we are unable to present more comprehensive hyperparameter tuning experiments than those presented later in this Appendix.

A.3 Llama Pretraining - Data Importance

In this section, we study the data points identified as easy and challenging by LFR when pretraining with the SlimPajama dataset. Listing A.3 provides an example of a code snippet from Github classified as easy by LFR, and discarded in the Focus stage of the Llama model training. Listing A.3 provides an example of a data sample retained from the Github cluster. Note that this code is more complex, presents an opportunity to the model to improve its coding capabilities as opposed to the variable declarations in Listing A.3.

Listing 1: Code snippet classified as easy by LFR, primarily consisting of variable declarations. As seen from the code, it contributes minimally to enhancing the model's coding capabilities.

Parameter	Value
Context Length	1024
Embedding Dimension	(768, 1024, 1280, 1600)
Total Iterations	40000
Effective Batch Size	512
Batch Size	(16, 16, 8, 4)
Gradient Accumulation Steps	(32, 32, 64, 128)
Block Size	1024
Weight Decay	1.00E-01
Adam β_1	0.9
Adam β_2	0.95
Warmup Iterations	2000
Minimum Learning Rate	6.00E-05
Maximum Learning Rate	6.00E-04
Learning Rate Schedule	Linear
Learning Rate Decay Iterations	40000
GPUs	(4xMI100, 4xMI210, 4xMI210, 4xMI210)

Table 4: Pretraining hyperparameters for the GPT-2 124M-1.5B parameter models. Parameters with multiple values (e.g. Embedding dimensions, batch size, gradient accumulation steps, and GPUs) specified in brackets are for the 124M, 345M, 774M, and 1.5B parameter models respectively.

```
package frclibj;

import edu.wpi.first.wpilibj.
    Timer;

public class TrcDbgTrace
{
    public static final String
        ESC_PREFIX          = "\u001b
        [";
    public static final String
        ESC_SUFFIX          = "m";
    public static final String
        ESC_SEP             = ";";

    public static final String
        SGR_RESET           = "0";
    public static final String
        SGR_BRIGHT          = "1";
```

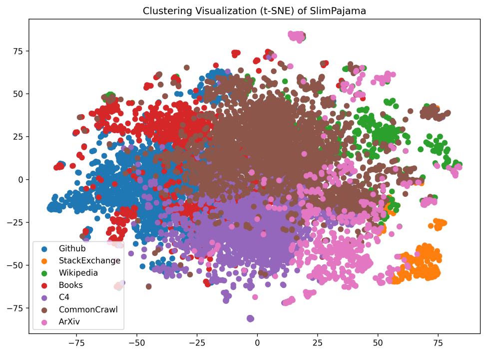

Figure 1: Clustering the data embeddings from the SlimPajama dataset using the Llama-300M model at the 50k training step.

```
    public static final String
        SGR_DIM             = "2";
    public static final String
        SGR_ITALIC          = "3";
    public static final String
        SGR_UNDERLINE       = "4";
    public static final String
        SGR_BLINKSLOW       = "5";
    public static final String
        SGR_BLINKFAST       = "6";
    public static final String
        SGR_REVERSE         = "7";
    public static final String
        SGR_HIDDEN          = "8";
    public static final String
        SGR_CROSSEDOUT      = "9";

    public static final String
        ESC_NORMAL          =
        ESC_PREFIX;
}
```

Listing 2: Code snippet classified as challenging by LFR. This code consists of a function which executes an Oracle query and returns a scalar value. As seen from the code, it contributes significantly to enhancing the model's coding capabilities as compared with Listing A.3.

```
/// <summary>
/// Executes an Oracle query that
    returns a single scalar value
    as the result.
/// </summary>
/// <param name="commandText">The
    Oracle query to execute</
   param>
```

```
/// <param name="parameters">
    Optional parameters to pass to
     the query</param>
/// <returns>The result of the
    query as an object</returns>
public object QueryValue(string
    commandText, IEnumerable
    parameters)
{
    object result;

    if (String.IsNullOrEmpty(
        commandText))
    {
        throw new
            ArgumentException("
            Command text cannot be
             null or empty.");
    }

    try
    {
        ensureConnectionOpen();
        var command =
            createCommand(
            commandText,
            parameters);
        result = command.
            ExecuteScalar();
    }
    finally
    {
        ensureConnectionClosed();
    }

    return result;
}
```

Similarly, we also provide examples of question-answer pairs from StackExchange which were discarded and retained in the Focus stage of the Llama pretraining in Listings A.3 and A.3 respectively.

Listing 3: Question-answer pair from StackExchange classified as easy by LFR. The question revolves around a process in PayPal which does not contribute as much to the answering capability or world knowledge of the model.

```
Q: PayPal IPN $_POST['txn_id']
    not set. I'm using the PayPal
    sandbox to do a subscribe
    button, and then when I get
    the IPN response for a
    subscription or a subscription
     cancellation $_POST['txn_id']
     is never set.
So I don't know how to identify
    transactions to only accept
    unique ones.
Thanks!
EDIT: for example, all the info
    that I have in POST for a
    subscr_cancel are:
amount1, amount3, address_status,
     subscr_date, payer_id,
    address_street, mc_amount1,
    mc_amount3, charset,
    address_zip, first_name,
    reattempt,
    address_country_code,
    address_name, notify_version,
    subscr_id, custom,
    payer_status, business,
    address_country, address_city,
     verify_sign, payer_email,
    btn_id, last_name,
    address_state, receiver_email,
     recurring, txn_type,
    item_name, mc_currency,
    residence_country, test_ipn,
    period1, period3,
    correlation_id.

A: According to Table 2. Summary
    of subscription variables:
For subscription variables, the
    transaction ID (txn_id) is
    only available for USD Payment
     and Multi-Currency Payment
    transaction types (txn_type).

As expected, PayPal will not send
     the txn_id to your IPN for
    the transaction type,
    subscr_cancel, and will only
    send txn_id if the transaction
     type is subscr_payment.

For further explanation on which
    variables are sent to your IPN
     URL based on your transaction
    , please check out IPN and PDT
     Variables.

Have you checked $_REQUEST['
```

```
    txn_id'] as this may be sent
    to your server via GET.
```

Listing 4: Question-answer pair from StackExchange classified as challenging by LFR. The question revolves around solving an ODE which contributes more to the learning of the model than Listing A.3.

```
Q: Passing additional iteration-
    dependent inputs to ode45
I'm trying to solve a
    differential equation using
    the ode45 function. Consider
    the following code,
[t1,X2] = ode45(@(t,x)fun(t,x,C1,
    C2,C3,C4),t0,X01);

where parameters C1, C2, C3, and
    C4 are column vectors, which
    should be available to the
    function that ode45 is
    referring to (fun.m).
I want the values to change after
     every iteration, so for
    example, at the beginning the
    entry of C1 I want is C1(1),
    in the next iteration it's C1
    (2), etc.
How can I implement that?

A: You may have noticed that the
    official docs are not too
    helpful in this scenario (as
    they pretty much force you to
    use global variables - which
    is doable, but discouraged).
Instead, I'll show you how this
    can be done with classes and
    function handles. Consider the
     following:

classdef SimpleQueue < handle
  %SIMPLEQUEUE A simple FIFO data
      structure.

  properties (Access = private)
    data
    position
  end

  methods (Access = public)
    function obj = SimpleQueue(
        inputData)
      %SIMPLEQUEUE Construct an
          instance of this class
      obj.data = inputData;
      rewind(obj);
    end % constructor

    function out = pop(obj,
      howMany)
      %POP return the next
          howMany elements.
      if nargin < 2
        howMany = 1; % default
            amount of values to
            return
      end
      finalPosition = obj.
          position + howMany;
      if finalPosition > numel(
          obj.data)
        error('Too many elements
            requested!');
      end
      out = obj.data(obj.position
          + 1 : obj.position +
          howMany);
      obj.position =
          finalPosition;
    end % pop

    function [] = rewind(obj)
      %REWIND restarts the
          element tracking
      % Subsequent calls to pop()
           shall return elements
          from the beginning.
      obj.position = 0;
    end % rewind
  end % methods
end % classdef

How to use this? Simple:
C1q = SimpleQueue(C1);
C2q = SimpleQueue(C2);
C3q = SimpleQueue(C3);
C4q = SimpleQueue(C4);

[t1,X2] = ode45(@(t,x)fun(t,x,
    @C1q.pop,@C2q.pop,
@C3q.pop,@C4q.pop),t0,X01);

As you can see, inside fun we use
    C1q() instead of C1.
```

A.4 Sensitivity Study

In this section, our goal is to understand the effects of more aggressive focus, revision, and learning strategies than the training strategy presented in the paper. Here, we vary the values of hyperparameters p_1, s_1, p_2, p_3, and $reps$ and study the effects on the downstream task perplexity. Note that the GPT-2 models used a four phase training process. Specifically, we aim to answer the following two questions using the GPT-2 models:

1. What is the impact of not reintroducing the discarded data samples?
2. What is the impact of the degree of pruning in Phases 2 and 4?

To answer the first question, we pretrain a 124M parameter GPT-2 model without the reintroduction of data blocks in Phase 3, and use the reduced subset from Phase 2 for the rest of the training. Then, we finetune for downstream language modeling tasks similarly and compared the perplexities using LFR in Table 5. This training strategy which removes Phase 3, is labeled as `no-reintro`. Next, to answer the second question, we pretrain a 124M parameter GPT-2 model using LFR but increase the degree of pruning in Phase 2 from 50% to 70%, i.e., reduce the training subset to 30% of the original size. This aggressive training strategy is labeled as `aggr-2`.

We observe that both aggressive training strategies do not work as well as the original method. However, we continue to explore more automated ways of deciding the training schedule for different model families as part of our future work.

Model	WikiText-2	WikiText-103	LAMBADA	1BW
`no-reintro`	23.24	25.76	17.27	36.02
`aggr-2`	23.91	27.00	21.11	38.62
LFR	19.81	22.49	16.61	32.27

Table 5: Downstream task perplexities with more aggressive training strategies.

A.5 Analysis on Dropped and Retained Data Blocks - GPT-2

In this section, our goal is to characterize the data points retained and dropped during pretraining by LFR in Phases 2 and 4 across the training time and model size. Specifically, we aim to answer the following questions:

1. What types of data blocks are learned earlier in the training process compared to those learned later?
2. Are similar data blocks considered learned and dropped in Phases 2 and 4?
3. Are the dropped data blocks similar across model sizes?
4. Are the data blocks dropped similar to those retained at any given training phase?

To answer the first question, we printed out the texts dropped and retained at different training phases. Tables 9 and 11 show text blocks dropped in Phases 2 and 4 by the 345M and 124M parameter models, while Tables 10 and 12 show data blocks retained. By reading through the texts, we notice that the model first learned conversations and personal anecdotes, before being able to retain factual information. We provide a more detailed analysis of the learning process in Section A.6.

In order to answer questions 2-4, we recorded only the IDs of dropped data blocks during Phases 2 and 4 for both the GPT-2 124M and GPT-2 345M models, totaling 4 lists of dropped IDs. We then load the recorded data blocks and embed them into a higher dimensional space using the GPT-2 tokenizer. Considering that there is a total of 8.7M data blocks (9B tokens divided into blocks of 1024 tokens), we cluster the embeddings using k-means clustering with $k = 270$ to reduce the analysis space and complexity. Finally, for each model, we compute the cosine similarity for all combinations of the embeddings of dropped data blocks across training phases and visualize them using a heatmap. These heatmaps plot the cosine similarity values (ranging between 0 and 1) such that lighter values (closer to 1) indicate higher similarity.

Figure 2 shows the similarity of dropped data blocks across the time scale (Phase 2 shown on the X-axis and Phase 4 shown on the Y-axis) for the 124M (left) and 345M (right) parameter models. We find that there is a higher similarity in the data points dropped by the 124M parameter model in Phases 2 and 4 than in the case of the 345M parameter model (mean, variance, and standard deviation are provided in Table 6). This behavior signals that the lower capacity of the 124M parameter model inhibits its learning process in Phase 3, such that it finds similar points confusing in Phases 2 and 4. In contrast, the 345M parameter model learns the data

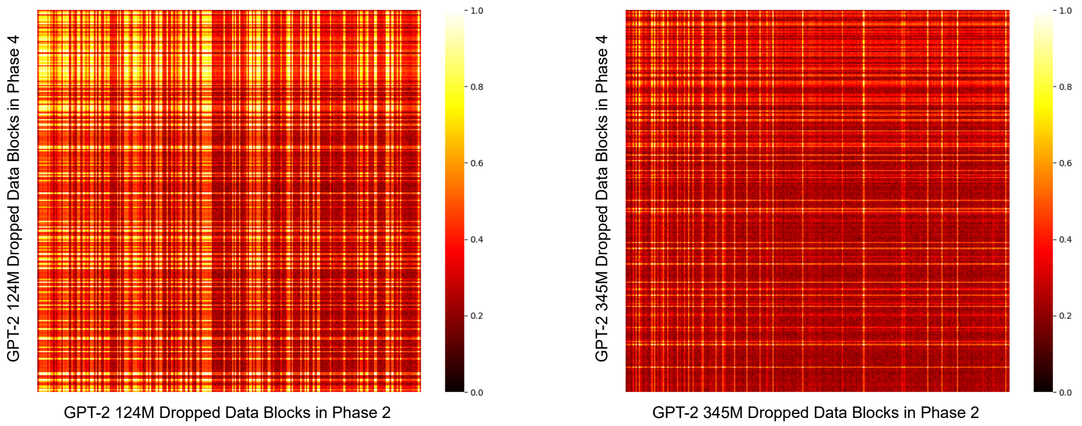

Figure 2: Cosine similarity heatmaps for dropped data blocks during phases 2 and 4 of pretraining for the GPT-2 124M (right) and 345M (left) models. The smaller model displays greater similarity in dropped data blocks over time (lighter color), indicating that it remained uncertain about similar data points than the larger model.

blocks it found confusing in Phase 2 by focusing on them, and moves on to learning new complex blocks by Phase 4.

We conduct a similar study in order to characterize the similarity in data blocks across model sizes. Figure 3 plots the cosine similarity heatmap for the data blocks dropped by the 124M parameter model (X-axis) and those dropped by the 345M parameter model (Y-axis) in Phase 2. The mean, variance, and standard deviations of the cosine similarity are 0.38, 0.15, and 0.023, respectively. This indicates that the data blocks found easy and dropped in Phase 2 by both models display a moderate level of similarity, but also differ significantly.

Finally, we observe the cosine similarity of data blocks dropped and retained during phase 4 for the 124M (left) and 345M (right) parameter models in Figure 4. The mean, standard deviation, and variance are detailed in Table 7. The smaller model displays greater similarity (lighter values in the heatmap) between the dropped and retained blocks than the larger model. We hypothesize that the larger model can perform reasonably well across similar data points, but struggles with very different complex blocks by the fourth training phase. In contrast, the smaller model does not display the same high-level of understanding (similar perplexity values) on related data blocks.

To summarize, **data block importance varies across training time, and across model sizes**. Therefore, static data selection techniques (Tirumala et al., 2023; Abbas et al., 2023; Kaddour, 2023; Xie et al., 2023) which select a fixed subset

Model	Mean	Std	Variance
GPT-2 124M	0.45	0.20	0.04
GPT-2 345M	0.30	0.12	0.01

Table 6: Mean, standard deviation (std), and variance of cosine similarity matrices for dropped data blocks across time scale (Phase 2 and Phase 4) for the GPT-2 124M and 345M models.

Model	Mean	Std	Variance
GPT-2 124M	0.44	0.21	0.046
GPT-2 345M	0.32	0.13	0.018

Table 7: Mean, standard deviation (std), and variance of cosine similarity matrices for dropped and retained data blocks in Phase 4 of pretraining for the GPT-2 124M and 345M models.

to train for the entire training duration for all model architectures do not adapt to the changing training dynamics of LLMs. Based on our analysis in Figure 3 and 2, different data blocks are found difficult by models of different capacities at different training instants, driving the need for dynamic data selection methods like LFR. We detail further analysis on the selected and discarded data blocks and demonstrate how models initially focus on learning conversational and anecdotal data, before proceeding to learn factual data in Appendix A.6.

A.6 Extended Analysis on Dropped and Retained Data Blocks for GPT-2

In this section, we expand on the ablation study in Section A.5 in order to better characterize the data

Model	Mean	Std	Variance
GPT-2 124M	0.42	0.19	0.04
GPT-2 345M	0.40	0.18	0.03

Table 8: Mean, standard deviation (std), and variance of cosine similarity matrices for dropped and retained data blocks in phase 2 of pretraining for the GPT-2 124M and 345M models.

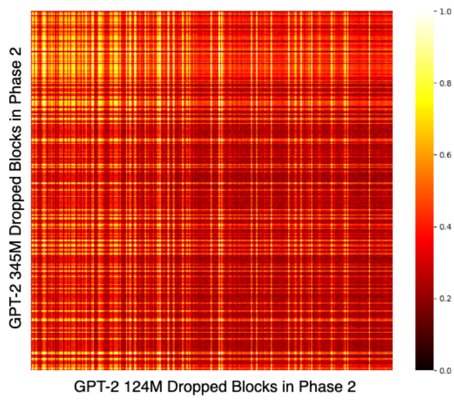

Figure 3: Cosine similarity heatmap for data blocks dropped during Phase 2 of GPT-2 124M and 345M pretraining shows moderate similarity, indicating different data points are considered easy by each model.

Become a fan of Slate on Facebook. Follow us on Twitter.The first time I crocheted a soccer ball was on the occasion of the 2010 World Cup. It was being held on the continent of Africa, and I thought the African Flower hexagon motif was the perfect vehicle for a crochet soccer ball celebrating the continent's first time hosting the World Cup: This time around, instead of using all 9000 of my favorite colors, I limited myself to the colors of the flags of the thirty-two countries that had made it to the final rounds of the World Cup competition, and I did my best to incorporate the designs of their flags into the thirty-two hexagons and pentagons of a soccer ball.
ML-77 Missile Launcher: Based on existing technology, the ML-77 is a rapid-fire missile launcher using seeking projectiles. Each projectile features a friend-or-foe recognition system, ensuring it will find a hostile target even if the user's aim is not completely accurate. The locking mechanism of the ML-77 allows the shooter to ignore cover and line of sight when shooting at locked on enemies, though an attack roll is still required. Locking on to an enemy requires a move action when the enemy is in line of sight and lasts for the rest of the encounter, or until a new target is locked.

Table 9: Examples of text dropped by the 345M model in phase 2 (top) and phase 4 (bottom).

blocks considered easy / hard.

Tables 9 and 11 provides examples of text blocks dropped in Phases 2 and 4 by the 345M and 124M parameter models respectively. Similarly, Tables 10 and 12 provide examples of data blocks retained by the models in Phases 2 and 4. We printed out and went over all the text dropped and retained in both Phases, and notice that text considered easy in phase 2 was more conversational, and those considered easy in phase 4 were more factual. This might indicate that the model first learned conversations and personal anecdotes, before being able to retain factual information. These findings are further corroborated by the examples of data retained in both phases. We are working on further analysis across different model families and sizes to strengthen this understanding.

Next, we continue the analysis of the cosine similarity heatmaps evaluated across training time and model parameter scales presented in Section A.5. Here, we answer the following questions:

1. Are there similarities in the data blocks considered easy and dropped in Phase 4 of training of the 124M parameter model with those considered easy and dropped by the 345M parameter model in Phase 2?

2. Are the data blocks dropped similar to those retained at any given training phase? Note that Section A.5 presented this analysis only for Phase 4 of the 124M and 345M parameter models in Figure 4.

Figure 5 depicts the cosine similarity heatmap

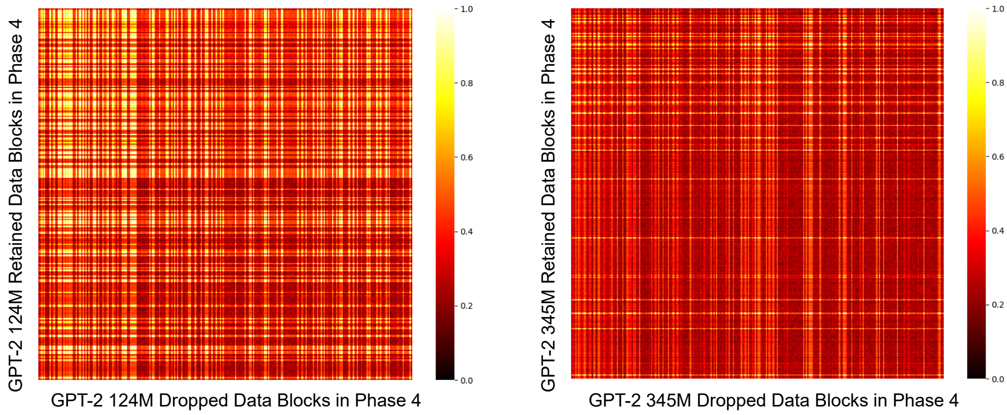

Figure 4: Cosine similarity heatmaps for dropped and retained data blocks during Phase 4 of pretraining for the GPT-2 124M (right) and 345M (left) models.

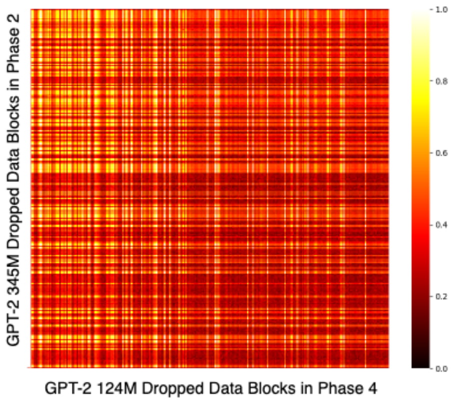

Figure 5: Cosine similarity heatmap for dropped data blocks during Phase 4 of GPT-2 124M and Phase 2 of the 345M model.

for the data blocks dropped by the 124M parameter model in Phase 4 (X-axis) with those dropped by the 345M parameter model in Phase 2 (Y-axis). The mean, standard deviation, and variance of the similarity are 0.43, 0.18, and 0.03 respectively. In contrast, the mean cosine similarity of data blocks dropped in Phase 2 of pretraining of both the models was 0.38 (Section A.5 and Figure 3). This indicates that the smaller model "catches up" with the knowledge accumulated by the larger model, and considers similar block easy in Phase 4 as those considered easy by the larger model in Phase 2.

Next, we plot the cosine similarity heatmap for the dropped and retained data blocks in Phase 2 for the 124M (left) and 345M (right) parameter models. The mean, variance, and standard deviations of the similarity are shown in Table 8. Observing the mean similarity value and heatmap in Table 7 and Figure 4, we find that the cosine similarity for dropped and retained data blocks is higher in Phase 2 than Phase 4 in case of the 345M parameter model. In contrast, the value remains high in both Phases for the 124M parameter model. This finding indicates that both the smaller and larger model start the training by being confused about similar data blocks. However, the larger capacity of the 345M parameter model allows it to learn the dataset well in Phases 2 and 3, and move on to more complex data blocks in Phase 4 (thus reducing the mean similarity in Phase 4). The smaller model continues remaining unsure about similar data blocks. Since we observed that the smaller model "catches up" with the training of the larger model (in Figure 5), we hypothesize that the smaller model will eventually display similar behaviour as the larger model once trained for longer iterations.

References

a. Arc Challenge Dataset. https://huggingface.co/datasets/allenai/ai2_arc.

b. Arc Easy Dataset. https://huggingface.co/datasets/allenai/ai2_arc.

a. BookCorpus Dataset. https://huggingface.co/datasets/bookcorpus/bookcorpus.

b. BoolQ Dataset. https://huggingface.co/datasets/google/boolq.

HellaSwag Dataset. https://huggingface.co/datasets/DatologyAI/hellaswag.

OpenBookQA Dataset. https://huggingface.co/datasets/allenai/openbookqa.

PIQA Dataset. https://huggingface.co/datasets/ybisk/piqa.

WikiText Dataset. https://huggingface.co/datasets/Salesforce/.

WinoGrande Dataset. https://huggingface.co/datasets/allenai/winogrande.

Amro Abbas, Kushal Tirumala, Dániel Simig, Surya Ganguli, and Ari S. Morcos. 2023. Semdedup: Data-efficient learning at web-scale through semantic deduplication. *Preprint*, arXiv:2303.09540.

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. 2018. Unsupervised neural machine translation. In *International Conference on Learning Representations*.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. *Preprint*, arXiv:1312.3005.

Jean Kaddour. 2023. The MiniPile Challenge for Data-Efficient Language Models. *Preprint*, arXiv:2304.08442.

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

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany. Association for Computational Linguistics.

Kushal Tirumala, Daniel Simig, Armen Aghajanyan, and Ari S. Morcos. 2023. D4: Improving LLM Pretraining via Document De-Duplication and Diversification. *Preprint*, arXiv:2308.12284.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy Liang. 2023. Data selection for language models via importance resampling. *Preprint*, arXiv:2302.03169.

Unofficial reports claimed the car was powered by a 95kW 1.5-litre non-turbo petrol engine but Tada didn't confirm. When asked what powers the S-FR Tada revealed he was considering three choices. "When you see the S-FR concept I suppose you imagine it is a 1.5-litre car but nowadays I can choose many kind of engines," he explained. "Downsized turbo, 1.5-litre naturally aspirated and something additional as well. Now we are thinking which one is the best engine for a small sports car." Tada also admitted that the company is unlikely to turn to a partner like it did with Subaru for the 86/BRZ or the new 'big brother' sports car with BMW.
In April, MYIR released a Linux-powered MYS-6ULX single board computer, which was notable for being available in two different versions using NXP's low power, Cortex-A7 i.MX6 UltraLite (UL) or the more affordable, and almost identical i.MX6 ULL SoC. Now, MYIR has released an "MYB-6ULX Expansion Board" designed to stack onto either model. The $21.20 accessory adds a second 10100 Ethernet port to the MYS-6ULX, as well as new CAN, RS485, audio, micro-USB, RTC, and camera functions. MYB-6ULX Expansion Board with MYS-6ULX (left) and detail view (click images to enlarge). The MYB-6ULX Expansion Board has the same 70 x 55mm dimensions as the MYS-6ULX, which is available in two models: The i.MX6 UL based MYS-6ULX-IND has -40 to 85°C support instead of 0 to 70°C, and the i.MX6 ULL based MYS-6ULX-IOT features a USB-powered WiFi radio. The 4-layer expansion board runs on 5V power, and shares the industrial temperature support of the IND model.

Table 10: Examples of text retained by the 345M model in Phase 2 (top) and Phase 4 (bottom).

In the book, the mythical California is ruled by Queen Califa and populated only with female warriors who brandish gold weapons. They even harness their animals in gold because it is the only mineral on the island. The legend of Califa and her island was well known among New World explorers. In 1536 when Hernán Cortéz arrived in Baja California, he believed he had landed on the legendary island. Over three hundred years later gold was discovered in California, making the legend partially true and earning the state its nickname: The Golden State.
Segregated Witness, defined by Bitcoin Improvement Proposal 141 (BIP141), was deployed using an activation mechanism (BIP9) that requires 95 percent of all miners (by hash power) to signal support for the upgrade within the span of a two-week difficulty period. That's at least 1916 blocks within 2016 blocks, to be exact. This threshold has just been reached. While the current difficulty period will not end until tomorrow, all blocks in this difficulty period are signaling support for the upgrade so far. This now totals over 1916 of them.

Table 11: Examples of text dropped by the 124M model in Phase 2 (top) and Phase 4 (bottom).

to the GUI installer. Most notably there is no support for configuring partition layout, storage methods or package selection. Please refer to the official documentation for details. Here you can find some useful information on creating and using kickstart files which can be used to perform advanced configuring without the need for the GUI installer. The message "Insufficient memory to configure kdump!" appears during install. This is a known issue which appears on systems with less than 2 GB RAM. This can be ignored. Content for both the i386 and x86_64 architectures is split into two DVDs. We have tried to get all basic server and basic desktop installs only from DVD-1. Make sure that you setup correctly the selinux context of the public key if you transfer it to a CentOS 6 server with selinux enabled.
Once you signed up, you can either click on the Todo tab or the sign in link to gain access to the application. Note that if you are selecting sign in in the same session in which you signed up, you will automatically sign in with the same account you used for signing up. If you are signing in during a new session, you will be presented with Azure AD's credentials prompt: sign in using an account compatible with the sign up option you chose earlier (the exact same account if you used user consent, any user form the same tenant if you used admin consent). If you try to sign-in before the tenant administrator has provisioned the app in the tenant using the Sign up link above, you will see the following error.

Table 12: Examples of text retained by the 124M model in phase 2 (top) and phase 4 (bottom).