GAFF: GPT-NeoX Active Forgetting Functionality

Linus Folkerts, Enric Balaguer Rodon, Sage Bergerson, Charles-Antoine D'Ornano, Nathan Herzhaft
University College London

Abstract

Large Language Models (LLMs) are the driving force behind much innovation in Natural Language Processing. However, although the availability of open-source pre-trained language models has increased, working directly with LLM pre-training itself in an academic environment remains challenging due to computational and technical resource constraints. For academic users, the LLM pre-training process is opaque and complex. Inspired by this in general, and the challenge of designing LLM pre-training to produce models that can quickly adapt to new languages in particular, the GPT-NeoX Active Forgetting Functionality (GAFF) was developed. GAFF is a simple Active Forgetting mechanism, developed in and compatible with an academic computing environment, that works with all models built with GPT-NeoX. Specifically, GAFF resets embedding parameters every k updates during pre-training, encouraging models to learn new embeddings in a smaller number of updates. Preliminary experiments with Pythia validate the functionality of the method and demonstrate an approach for designing and understanding pre-training processes in constrained computing environments.

1 Introduction

The original research has been to extend the work of (Chen et al., 2024) to autoregressive language models to answer similar questions about pre-trained language models and Active Forgetting in a new context: **Can pre-training with Active Forgetting increase plasticity in pre-trained language models? With increased plasticity, can pre-trained language models trained with Active Forgetting reduce the fine-tuning time and compute resources required?**

Focusing on masked language models, Chen et al. follow a design in which their transformer body is pre-trained with token embeddings being reset via Active Forgetting at regular intervals, followed by separate task-adapt and language-adapt phases in which the transformer body is fine-tuned on English task data while the token embeddings are fine-tuned on new, unlabelled language data.

For simplicity, this research planned to adapt these experiments to only include the pre-training and language-adapt phases, and exclude the task-adapt phase. To determine the effects of Active Forgetting, the convergence speed during the language-adapt phase would be measured, as opposed to task performance in the new language.

However, initial research and experimentation in a compute and support-constrained environment resulted in a move of focus. It became clear, that beyond Active Forgetting and its implications alone, there is a wider problem in need of attention: **How can academic users in a highly compute-constrained environment understand and shape large language model training pipelines and how can that guide the transfer of Active Forgetting to a new model architecture (autoregressive) and a new software stack (GPT NeoX)?**

As an answer to this question, *GPT-NeoX Active Forgetting Functionality (GAFF)* has been created. It shows how it is possible to approach complex and opaque training stacks to understand and shape them in general, while at the same time, allowing for the investigation of the transferibility of Active Forgetting in particular. The created method is general enough to pave the way for future implementations of curriculum learning and meta-learning approaches beyond Active Forgetting, highlighting its wide scope.

Both the method itself and the transfer of Active Forgetting, in general, could be partially validated. A complete service failure on the only

available compute platform Myriad over the last week of the project led to a severe limitation of validation capabilities. Through significantly extended effort, it was however possible to leverage commercial compute solutions and arrive at a minimum of benchmarking.

2 Related Work

2.1 Meta-learning

Meta-learning has become an influential approach in machine learning research to facilitate fast adaptation to new training data (Lake et al., 2015). In Natural Language Processing (NLP), meta-learning involves training models in a way that makes further downstream training quicker, often referred to as learning-to-learn.

(Finn et al., 2017) introduce model-agnostic meta-learning (MAML), a meta-learning algorithm compatible with any model trained using gradient descent. Using MAML, model parameters are trained such that a limited amount of gradient steps with a small amount of new training data produces good generalization on a given task, in essence, making the model easier to fine-tune. Similar to MAML, OpenAI's Reptile meta-learning algorithm (Nichol et al., 2018) simulates gradient descent within each task's parameter space, seeking an initialization for network parameters that allows for fine-tuning with a small amount of data from a new task.

Meta-learning in NLP has previously been used with transformer-based models and shown to improve their cross-lingual transferability (Chi et al., 2021), task adaptation (Hou et al., 2022), and performance in low-resource scenarios (Dou et al., 2019).

Active Forgetting induces a similar effect as meta-learning in the sense that both methods teach models how to learn more effectively.

2.2 Curriculum Learning

Curriculum learning involves progressively training models on simple to complex tasks, and can also enhance learning efficiency (Bengio et al., 2009). In NLP, this often involves training language models first on general patterns characteristic of the training data as a whole before moving on to specific patterns characteristic of a certain subdomain. Curriculum learning has proved to be particularly beneficial in adapting to domains with limited data (Shi et al., 2014), and can significantly reduce the training resources required for training transformer-based models (Garg et al., 2023; Platanios et al., 2019).

2.3 Language Model Training Tools

There are several existing toolkits and libraries designed to allow researchers and developers to customize, train, and understand language models.

Fairseq is a toolkit developed by Facebook AI Research (FAIR) (Ott et al., 2019) for training and evaluating sequence-to-sequence models in NLP tasks. Fairseq provides functionalities for modifying pre-training in LLMs, including encoder-only and encoder-decoder transformers, such as customization of architectures and modification of layer configurations, hidden dimensions, attention mechanisms, activation functions, dropout rates, and other hyperparameters.

Developed by Microsoft, DeepSpeed is a deep learning optimization library (Aminabadi et al., 2022) that can be used for exploring new model architectures, training methodologies, and optimization techniques.

AllenNLP (Gardner et al., 2018) is a research library built on PyTorch which supports use with both traditional NLP architectures and transformer-based models. AllenNLP can be used to define custom models with specific architectures and provides tools for hyperparameter tuning and grid search as well as utilities for model interpretability and analysis, such as attention visualization, saliency maps, and gradient-based attribution methods.

GPT-NeoX (Black et al., 2022), which is extensively used in this research, is an open-source language model training platform developed by EleutherAI. It makes extensive use of various optimization technologies and allows the distributed training of trillion-parameter-scale models.

2.4 Cross-lingual Generalization

Model generalization across languages demonstrates the ability of PLMs to capture universal patterns in language structure, grammar, and semantics that transcend specific languages.

2.4.1 Pre-training on Multiple Languages

Pre-training on multiple languages is known to cause per-language performance declines, especially for resource-poor languages (Pfeiffer et al., 2022). (Pfeiffer et al., 2021) use matrix factorization to capitalize on latent knowledge of spe-

cific languages used during pre-training stored in the existing embedding matrix to improve performance. (Langedijk et al., 2022) employ meta-learning to cross-lingual dependency parsing to learn a parameter initialization able to adapt to new languages quickly.

2.4.2 Pre-training on a Single Language

Cross-lingual generalization can be achieved by adapting English PLMs to additional languages. (Artetxe et al., 2020) pre-train a transformer-based MLM before adapting it to new languages by learning a new embedding matrix with the same objective while freezing all parameters in other layers. (Marchisio et al., 2022) demonstrate how new language-specific embeddings can be trained by building a shallow mini-model using a subset of the original model's parameters before recombining these models. For vision-and-language tasks, (Liu et al., 2023) freeze their model's classification head during fine-tuning with text embeddings, while randomly initializing other parameters, or resetting them to their pre-trained values. The use of adapters, layers placed after feedforward layers, have also been used to add model capacity to learn modular language representations (Pfeiffer et al., 2020, 2022; Ansell et al., 2022). (Hu et al., 2023) use meta-learning while training an ARLM to learn which tokens to up-weight by reweighting the language modelling loss for each token during online fine-tuning.

2.5 Active Forgetting

(Chen et al., 2024) demonstrate that resetting a subset of model parameters during pre-training can improve model plasticity and improve generalization.

2.5.1 Computer Vision

(Alabdulmohsin et al., 2021) experiment with different reinitialization methods for convolutional neural networks for image classification. (Taha et al., 2021) use an evolutionary algorithm which includes a reset hypothesis in which the parameter values are perturbed over multiple epochs. (Ramkumar et al., 2023) employ weight reinitialization to selectively forget and then relearn weight values, finding that they can learn more generalizable features.

2.5.2 Natural Language Processing

(Chen et al., 2022) combine factorization-based models and graph neural networks while forgetting specific node embeddings during training to truncate message passing between nodes and encourage new graph reasoning with the new nodes. (Chen et al., 2024) apply a similar forgetting mechanism to token embeddings. During pre-training on an English dataset, they reset the input token embedding layer every $k = 1000$ iterations with a RoBERTa base model. They achieve faster convergence during fine-tuning on new languages, indicating the model has an increased ability to learn new embeddings over a limited number of updates.

3 Transferring Active Forgetting to Autoregressive Models

(Chen et al., 2024) focus their research on maksed language model (MLM) pre-training with language-specific tokenizers, and use RoBERTa-base as their pre-training model, a 12-layer transformer-based language model (Liu et al., 2019). MLMs predict missing words in a sentence by masking them during training, and learning context from both preceding and subsequent words. In contrast, autoregressive language models (ARLMs) generate text sequentially, predicting the next word based solely on the preceding words. This fundamental difference in training and prediction mechanisms results in MLMs often being used for tasks requiring an understanding of bidirectional context, while ARLMs excel in text generation. This work aims to transfer Active Forgetting to ARLMs. ARLMs are highly influential in current NLP applications due to their effectiveness in generating coherent and contextually appropriate text and form the basis of models including GPT-4 (OpenAI et al., 2024), XL-Net (Yang et al., 2020), and Transformer-XL (Dai et al., 2019).

3.1 nanoGPT Experiments

To start off experiments and inform future work, Active Forgetting has first been implemented in a minimalistic model, nanoGPT [1]. nanoGPT operates on the same fundamental principles as larger and more complex training stacks but is radically less complex. The model trained is a simple transformer-based model, with 6 transformer blocks and 6 self-attention heads, and employs a technique known as weight tying (Press and Wolf, 2017), which involves linking the weights of the

[1] github.com/karpathy/nanoGPT

token embeddings and the language model head, allowing the model to share information between the encoding and prediction phases.

3.1.1 Experimental Setup

To study the effects of Active Forgetting, the convergence speed during fine-tuning on a new language on which the model had not been pre-trained was tracked for both our standard pre-trained model and our model pre-trained with Active Forgetting.

For pre-training a 500 MB subset of the CC-100 English dataset was used, and for fine-tuning a 20 MB subset of the CC-100 German dataset was used. [2] For pre-training and fine-tuning the standard model, the procedures defined in the nanoGPT GitHub repository were replicated, with customization of the main training loop in the train.py file via passing new variables into the associated config file to override default variables related to dataset paths. For the Active Forgetting model, an active_forgetting function was introduced to the model class and additional parameters were incorporated into the training process, including the parameter k (the number of updates that should occur between each Active Forgetting reset) and adjustments to the learning rate. Both models were trained using the same pre-training and fine-tuning configurations to isolate the effects of Active Forgetting.

3.1.2 Implementation

The active_forgetting function in the model class replicates the code used to initialize the embedding parameters, resetting all embedding parameters with a mean of 0.0 and standard deviation of 0.02. Inside the main training loop, the active_forgetting function is called every $k = 250$ updates. In the standard pre-training run, a cosine learning rate decay scheduler is used with a linear warmup. In the Active Forgetting pre-training run, a cyclic version of this scheduler is implemented. All experiments were run in Google Colab, making use of the NVIDIA V100 Tensor Core GPUs available through the platform. The pseudo code is presented in the appendix.

3.1.3 Results

To evaluate both models, curves of the difference between the loss at time t and the loss at time $t-1$ for all time steps t during both pre-training and fine-tuning were plotted, which provides insight into the rate of learning during training. The figures are presented in the appendix at the end of this report.

3.1.4 Discussion

During pre-training with Active Forgetting, nanoGPT struggled to recover from having its embedding parameters reset, which manifested as the loss returning nearly to its initial value after each reset. [3] Thus, by the end of pre-training with Active Forgetting, the model is still close to its initialized state.

Both models reached convergence around 230 iterations, with the Active Forgetting model demonstrating a more significant decrease in loss compared to the baseline model. However, given that the Active Forgetting model was close to its initialized state, this result could potentially be attributed to the Active Forgetting model essentially learning from scratch during fine-tuning. Given the successful implementation but inconclusive results, hypothesized to be due to the small size of nanoGPT, investigation on Active Forgetting in larger models began.

3.2 Extension to GPT-NeoX

The Pythia model suite was selected for the larger-scale implementation, a set of decoder-only autoregressive transformer-based models ranging from 14M to 12B parameters (Biderman et al., 2023), trained using GPT-NeoX. Predefined configurations for Pythia are included in GPT-NeoX, an open-source, ARLM created by EleutherAI, designed to replicate functionalities similar to OpenAI's GPT-3 (Black et al., 2022). GPT-NeoX is characterized by its focus on optimization and large-scale distributed training capabilities.

The Pythia model architecture and hyperparameters are close to those in (Brown et al., 2020) except for a few changes made based on advances in best practices for LLMs: fully dense layers, Flash Attention (Dao et al., 2022) during training to improve device throughput, rotary embeddings (Su et al., 2021) as positional embeddings, the parallelized attention, feedforward technique and model initialization methods from (Wang and Komatsuzaki, 2021) to improve training efficiency, and untied embedding and unembedding matrices to facilitate interpretability research (Belrose

[2] data.statmt.org/cc-100

[3] aizi.substack.com/p/how-does-gpt-3-spend-its-175b-parameters

et al., 2023). Using the Pythia models as a testbed, the goal was to develop a method to implement Active Forgetting for GPT-NEoX.

4 GPT-NeoX Active Forgetting Functionality (GAFF)

To extend Active Forgetting to arbitrary GPT-NeoX language model pre-trainings, the GPT-NeoX Active Forgetting Functionality (GAFF) has been created. Not only is it able to extend this research to far bigger models, it also provides a tool to shape opaque and complex training procedures in general.

4.1 Pre-Training System

The Pythia model family on which the research aims to validate the creaaed methodology on has been trained using GPT-NeoX is designed for use with multi-billion parameter models, employs several sophisticated optimization schemes, and requires a comprehensive system setup to install a wide range of requirements and dependencies.

To access the compute necessary for pre-training, which exceeded that of personal computers, the UCL High-Performance Computing cluster (HPC cluster) and the UCL Computer Science lab machines with Nvidia GeoForce RTX 3090 graphics cards (3090s) were used. However, the internal job scheduling system for the HPC cluster prevented code changes from being made in a timely manner, and a lack of root access, which was shared by the 3090s, meant installation of dependencies was only possible through containerization technologies which further increased iteration time, leading to an unsuitable environment for development. In addition, the provisioning of suitable containers for the usable technologies *itself*, was not possible in the provided environments and suitable environments had to be rented.

In hopes of setting up an initial standard training process that could later be transferred into a suitable format for the HPC cluster and 3090s, a machine with a GTX4090 graphics card was sourced on vast.ai [4], which allowed for full root privileges and instant job runs. After a phase of heavy engineering work preparing the complex dependency structure for GPT-NeoX, a standard pre-training for Pythia 14M was run using this setup. Having validated this training process, the next step was to translate the environment to be compatible with the HPC cluster and 3090s.

To overcome the limitation of no root user access, the pre-training process was run in a container, allowing for the inclusion of all requirements and dependencies. Docker, an OS-level virtualization to deliver software in packages (in which the dependency structure was defined by GPT-NeoX), [5] was incompatible with both the HPC cluster or 3090s, and the resources required to construct the container exceeded those available on personal computers. Thus, a virtual machine on the Google Cloud Platform [6] was rented to construct the Docker container, and this container was then translated into Apptainer, [7] a format compatible with the HPC cluster and 3090s.

When using the Apptainer container on the HPC cluster, the Intel C++ Compiler Suite [8] was marked as missing. This was resolved by changing the compilation process in Megatron, NVIDIA's framework of GPU optimized techniques for training transformer models at-scale, [9] to utilize a different compiler, Gnu C++ Compiler, [10] by injecting an environment variable into the container. After these adjustments, standard pre-training could be successfully run on the HPC cluster. This system was also compatible with the 3090s, although these machines did not have enough compute for a full pre-training run.

4.2 Active Forgetting Implementation

Similar to (Chen et al., 2024), GAFF implements AF by resetting the embedding layer of the model every $k = 1000$ updates. However, in contrast to previous Active Forgetting work and to provide a more general approach to modifying training pipelines that could potentially go beyond GPT-NeoX, GAFF was implemented in a way that does not need a modification of the training software stack itself.

The created *active_forgetting_run.py* Python module instead monitors a training run externally, stops it after k iterations, resets the embedding layer and optimiser state to then resume the training. This is made possible by loading a created

[4] vast.ai

[5] docker.com

[6] cloud.google.com

[7] apptainer.org

[8] intel.com/content/www/us/en/developer/tools/oneapi/dpc-compiler.html

[9] github.com/NVIDIA/Megatron-LM

[10] gcc.gnu.org

checkpoint and resetting the required weights in there while the training process is paused.

In addition to the embedding layer weights, the Adam optimizer state was also reset during each reset, as the parameter gradients were able to recover extremely quickly due to the Adam optimizer architecture. The Adam optimizer (Kingma and Ba, 2017) keeps track of the moving average of each parameter's gradient, allowing these values to resume almost instantly. Leveraging this pre-reset information almost entirely negated the effects of forgetting induced by resetting the embedding layer.

Optimizers like stochastic gradient descent (SGD) do not keep track of any moving averages and would not need to be reset. An optimizer like SGD could have been used with GPT-NeoX, however the Adam optimizer typically results in better performance. Implementing an SGD optimizer in the Apptainer would also have required significant additional engineering work.

Algorithm 1: AF mechanism GPT-NeoX

Input: n_{iter}: no. of iters. done, max_iter: no. of iters. to do, $P = (P_{embeds}, P_{body})$: params., k: interval between resets, get_lr: learning rate function, f: gradient function, $O = (O_{embeds}, O_{body})$: optimizer states, g: param. update function, $\mathcal{D}$: training data.

while $n_{iter} + k \leq max\_iter$ **do**
  load checkpoint ;
  for $i = 1, ..., k$ **do**
    $\alpha = get\_lr(n_{iter})$ (compute learning rate);
    $G = f(P, \mathcal{D})$ (compute gradients);
    $P_{embeddings} = g(G_{embeds}, P_{embeds}, O_{embeds}, \alpha)$;
    $P_{body} = g(G_{body}, P_{body}, O_{body}, \alpha)$;
    $n\_iter = n\_iter + 1$;
  end
  reset P_{embeds};
  reset O_{embeds};
  save checkpoint ;
end

4.3 Pre-training Experiments

To assess the effectiveness of GAFF, multiple Pythia 70m model pre-training runs have been planned, including different Active Forgetting schedules (different k values and different configurations of how much of the training process is covered through Active Forgetting). The research then aimed to run a standardized fine-tuning to establish a measure of the increased model plasticity as a function of the Active Forgetting schedule and method used.

Due to unexpected limitations in available compute, only a minimum of validation could be concluded.

4.3.1 Limitations

The Myriad HPC cluster, the only directly available source of necessary compute, began to experience suspected file system problems during the last week of the research project. The batch generation part of one step of the implemented training pipeline, which loads and distributes the training data for a particular batch, went from 0.34 seconds to 45.3 seconds (see 1), increasing the iteration time by more than 100 times and making it impossible to use Myriad for actual training runs.

Considerable effort was put into finding alternatives to the Myriad cluster. To have at least one baseline against which to compare the performance of Active Forgetting, a machine was rented from the vast.ai compute provider and set up for model training, representing severe additional implementation work (and monetary costs) but allowed a minimum of comparison. Even then, the breadth of validation and method assessment that could be provided fell far short of initial expectations and plans.

In addition to allowing a minimum of validation, the additional training run implementation on an external machine outside of Myriad confirmed that the fault was due to a failure of a Myriad system and not through a mistake in the training run configuration.

4.3.2 Pre-Training Data

As in (Chen et al., 2024), the 82G English subset of CC-100 was used for pre-training, constructed using URLs and paragraph indices from the CC-Net repository [11] by processing 2018 Common Crawl snapshots. CC-100 was chosen as the lan-

[11] github.com/facebookresearch/cc_net

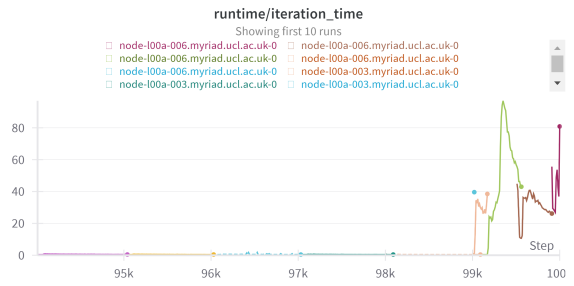

Figure 1: Faults on the Myriad system lead to a spike in training iteration time from 0.4s to 40s.

guage samples covered various topics and formats that were not task-specific, and several other language subsets generated using the same procedure were available for fine-tuning.

To use CC-100 with the Pythia models it was pre-processed to be compatible with GPT-NeoX. As a tokenizer, the Pythia model family tokenizer has been used.

4.3.3 Model Size Selection

Standard pre-training was initially tested with Pythia 350M on the CC-100 English dataset. However, the resulting model was severely undertrained, as it could not complete even one full epoch of training on the amount of compute available. Thus, the Llama-2-7b (Touvron et al., 2023) token-parameter ratio information [12] was used as a proxy to select a parameter count that would allow for sufficient pre-training convergence given the number of training tokens and amount of compute available. The configuration used in Llama-2-7b has become a standard in many recent open-source LLM projects, [13] [14] and the scaling between model and dataset size tends to follow a consistent trend (Kaplan et al., 2020).

Llama-2-7b used 2T training data tokens for a 7B parameter model, resulting in a ratio of 286 tokens per parameter. Based on the GPT-NeoX file dataset_token_count.py, [15] it was determined that the tokenized version of the CC-100 English dataset contained 24B tokens. According to this ratio, Pythia 70M was the optimal size, given that it would require 20B training tokens, just under the amount of data available.

[12] huggingface.co/meta-llama/Llama-2-7b

[13] github.com/juncongmoo/chatllama

[14] ai.meta.com/blog/code-llama-large-language-model-coding

[15] github.com/EleutherAI/gpt-neox/blob/main/tools/datasets/dataset_token_count.py

Configuration	Perplexity
Full Active Forgetting run	500.52
Baseline run	**153.9**

Table 1: The performance of the minimal range of validation runs measured as perplexity on the tiny-textbooks dataset.

4.3.4 Compute Usage

Before the presumed fault in Myriad file system, one full epoch of pre-training for Pythia 70M with the CC-100 English subset was estimated to require approximately 60 hours on 4 A100 GPUs available on the HPC cluster. To make this estimation, the number of tokens that could be processed during each training iteration was determined. As defined in the pythia-70m.yml file [16], Pythia 70M uses a sequence length of 2048 tokens, with 32 sequence inputs as batches per GPU in the pre-training loop. The pre-training loop completes 143K iterations on each GPU, with a single iteration processing a single batch. Therefore, 4 A100 GPUs could process $4 \times 2048 \times 32 \times 143\text{K} = 37.5\text{B}$ tokens, equivalent to about 1.56 epochs of the CC-100 English dataset. Thus, one full epoch would require 92K iterations per GPU, equating to 59 hours on each of the 4 A100 GPUs.

4.3.5 Pre-training Results

Due to the named Myriad service outage, the validation was limited to two pre-training runs: one on Myriad with Active Forgetting scheduled over the entire training with a reset every $k = 1000$ updates and one baseline run without pre-training and otherwise identical training parameters.

The full Active Forgetting run finished (reached 100k steps) before the service outage and therefore did not require the provisioning of additional compute through external providers and ran on a node with 4 A100 GPUs. However, a machine with 4 RTX6000 Ada GPUs hat to be sourced from vast.ai, to establish the performance baseline.

While initially fine-tuning runs were planned to validate the performance of different pre-training configurations, the mentioned limitations led to a simplified validation through the measured perplexity on the *tiny-textbooks* dataset (350M tokens) (Nam Pham, 2023).

The results are shown in table 1.

[16] github.com/EleutherAI/pythia/blob/main/models/70M/pythia-70m.yml

5 Discussion

The validation of implemented method is limited. The perplexity loss on the validation set is significantly higher on the Active Forgetting run compared to the baseline run. During the Active Forgetting run, the training process is not able to fully recover from each reset, as already seen with NanoGPT.

One hypothesis for why both nanoGPT and Pythia 70M struggle to recover from AF is the relative embedding parameter ratios combined with the size of both models. In nanoGPT, embedding parameters represent only about 19.8% of all parameters, but the total model size is only 11.4M. Although Pythia 70M has significantly more total parameters, embedding parameters account for 73.0%. Therefore, neither model has a large number of non-embedding parameters that are maintained during AF. In contrast, when using RoBERTa-base in (Chen et al., 2024), their model has 125M parameters, with 31.3% being embedding parameters, while in even larger models such as the full-sized GPT-3 model, embedding parameters account for less than 1%. Therefore, it is possible that a model must have an appropriate embedding parameter ratio for its size to be able to recover from AF.

Like planned, more training configurations and fine tuning would need to be tested to establish a complete picture on the performance of implemented method. This applies in particular to the k value which could have been set to high for the 70m training run, leading the model to learn "too many new" patterns between each reset.

Additionally, the recovery phase of every reset showed a plateau in loss during the Active Forgetting as seen in Fig. 3. The general performance of GAFF and patterns like this in particular can only be assessed through pre-training runs. The service outage of Myriad making this impossible was truly frustrating and disheartening for the team involved in the project.

6 Conclusions

GAFF, a simple Active Forgetting mechanism, was developed within and is compatible with an academic computing environment, and can be used to implement and customize AF pre-training in all models built using GPT-NeoX. More importantly, however, it can be used to better understand and shape conventionally opaque pre-training processes.

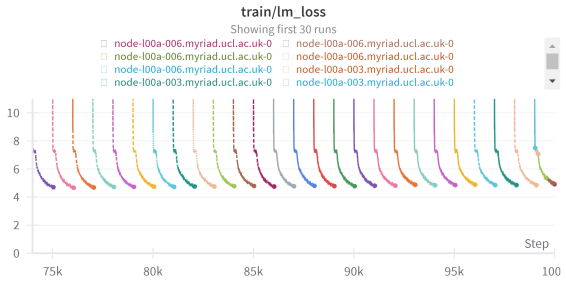

Figure 2: Training loss for the Active Forgetting run.

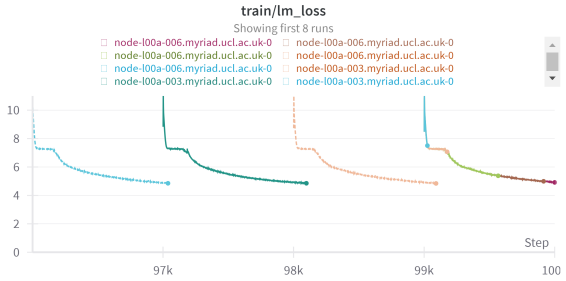

Figure 3: During the Active Forgetting run a plateau is visible after every embedding layer reset.

Active Forgetting itself could be extended in several possible directions, including determining the embedding parameter ratio thresholds required for ARLM models of different sizes to recover from AF during pre-training, various adjustments to the hyperparameters of GAFF or the timing of its introduction during pre-training, application to other GPT-NeoX models, and full replication of the experiments in (Chen et al., 2024) with an ARLM.

At the same time, the method could be used to investigate modifications to pre-training processes in a more general way, for example by using it to introduce a more granular form of curriculum learning.

But most importantly, the validation of proposed method could and should be extended, once Myriad is working again or other sources of compute become available.

7 Acknowledgements

We would like to thank Yihong Chen for the inspiration and guidance she provided to us throughout this project.

References

Ibrahim Alabdulmohsin, Hartmut Maennel, and Daniel Keysers. 2021. The impact of reinitialization on generalization in convolutional neural networks. *arXiv preprint arXiv:2109.00267*.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. Deepspeed inference: Enabling efficient inference of transformer models at unprecedented scale.

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulic. 2022. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796.

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. 2020. On the cross-lingual transferability of monolingual representations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4623–4637.

Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, Igor Ostrovsky, Lev McKinney, Stella Biderman, and Jacob Steinhardt. 2023. Eliciting latent predictions from transformers with the tuned lens.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. Gpt-neox-20b: An open-source autoregressive language model.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Greg Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Yihong Chen, Kelly Marchisio, Roberta Raileanu, David Ifeoluwa Adelani, Pontus Stenetorp, Sebastian Riedel, and Mikel Artetxe. 2024. Improving language plasticity via pretraining with active forgetting.

Yihong Chen, Pushkar Mishra, Luca Franceschi, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2022. Refactor gnns: Revisiting factorisation-based models from a message-passing perspective. In *Advances in Neural Information Processing Systems*.

Zewen Chi, Heyan Huang, Luyang Liu, Yu Bai, and Xian-Ling Mao. 2021. Cross-lingual language model meta-pretraining.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness.

Zi-Yi Dou, Keyi Yu, and Antonios Anastasopoulos. 2019. Investigating meta-learning algorithms for low-resource natural language understanding tasks.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia. Association for Computational Linguistics.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. 2023. What can transformers learn in-context? a case study of simple function classes.

Zejiang Hou, Julian Salazar, and George Polovets. 2022. Meta-learning the difference: Preparing large language models for efficient adaptation. *Transactions of the Association for Computational Linguistics*, 10:1249–1265.

Nathan Hu, Eric Mitchell, Christopher D. Manning, and Chelsea Finn. 2023. Meta-learning online adaptation of language models.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Anna Langedijk, Verna Dankers, Phillip Lippe, Sander Bos, Bryan Cardenas Guevara, Helen Yannakoudakis, and Ekaterina Shutova. 2022. Meta-learning for fast cross-lingual adaptation in dependency parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8503–8520, Dublin, Ireland. Association for Computational Linguistics.

Chen Liu, Jonas Pfeiffer, Anna Korhonen, Ivan Vulic, and Iryna Gurevych. 2023. Delving deeper into cross-lingual visual question answering. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2408–2423.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Kelly Marchisio, Patrick Lewis, Yihong Chen, and Mikel Artetxe. 2022. Mini-model adaptation: Efficiently extending pretrained models to new languages via aligned shallow training. *arXiv preprint arXiv:2212.10503*.

Nam Pham. 2023. tiny-textbooks (revision 14de7ba).

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang,

Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

Jonas Pfeiffer, Naman Goyal, Xi Lin, Xian Li, James Cross, Sebastian Riedel, and Mikel Artetxe. 2022. Lifting the curse of multilinguality by pre-training modular transformers. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3479–3495.

Jonas Pfeiffer, Ivan Vulic, Iryna Gurevych, and Sebastian Ruder. 2020. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Ivan Vulic, Iryna Gurevych, and Sebastian Ruder. 2021. Unks everywhere: Adapting multilingual language models to new scripts. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10186–10203.

Emmanouil Antonios Platanios, Otilia Stretcu, Graham Neubig, Barnabas Poczos, and Tom M. Mitchell. 2019. Competence-based curriculum learning for neural machine translation.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models.

Vijaya Raghavan T Ramkumar, Elahe Arani, and Bahram Zonooz. 2023. Learn, unlearn and relearn: An online learning paradigm for deep neural networks. *Transactions on Machine Learning Research*.

Yangyang Shi, Martha Larson, and Catholijn Jonker. 2014. Recurrent neural network language model adaptation with curriculum learning. *Computer Speech Language*, 33.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *Computing Research Repository*, arXiv:2104.09864v4.

Ahmed Taha, Abhinav Shrivastava, and Larry S Davis. 2021. Knowledge evolution in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12843–12852.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ben Wang and Akira Komatsuzaki. 2021. Gpt-j-6b: A 6 billion parameter autoregressive language model.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. Xlnet: Generalized autoregressive pretraining for language understanding.

8 Appendix

Algorithm 2: Active Forgetting mechanism nanoGPT

Input: n_{iter}: no. of iters. so far, max_iter: no. of iters. to do, $P = (P_{embeds}, P_{body})$: model params., k: interval between resets, get_lr: learning rate function, f: gradient function, $O = (O_{embeds}, O_{body})$: optimizer states, g: param. update function, $\mathcal{D}$: training data.

while $n_{iter} \leq max\_iter$ **do**
  if $n_{iter} \bmod k = 0$ **then**
    reset P_{embeds};
  end
  $\alpha = get\_lr(n_{iter})$ (compute learning rate);
  $G = f(P, \mathcal{D})$ (compute the gradients);
  $P_{embeds} = g(G_{embeds}, P_{embeds}, O_{embeds}, \alpha)$;
  $P_{body} = g(G_{body}, P_{body}, O_{body}, \alpha)$;
  $n\_iter = n\_iter + 1$;
end

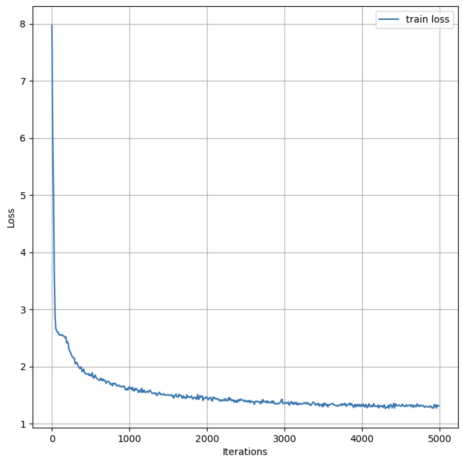

Figure 4: Standard nanoGPT pre-training loss.

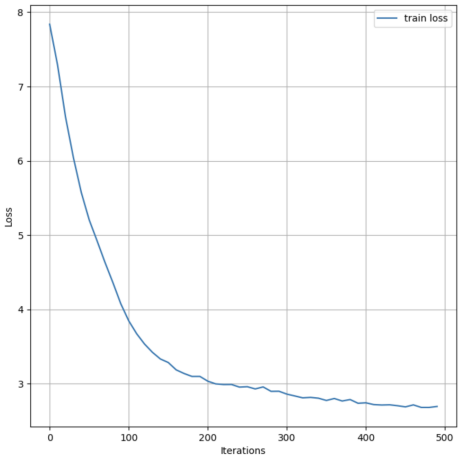

Figure 5: Standard nanoGPT fine-tuning loss.

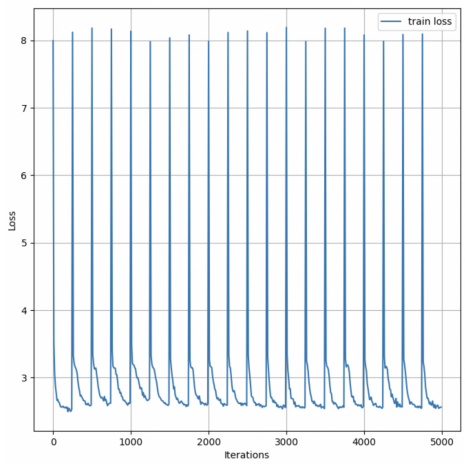

Figure 6: Active Forgetting nanoGPT pre-training loss.

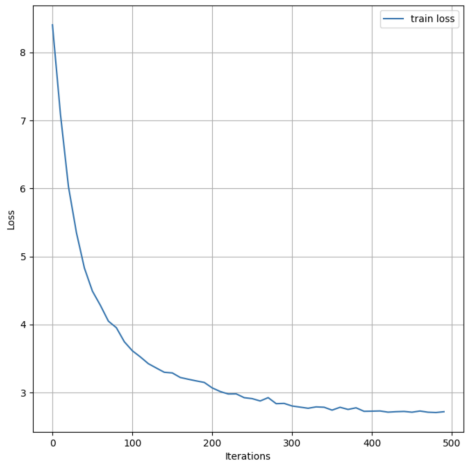

Figure 7: Active Forgetting nanoGPT fine-tuning loss.

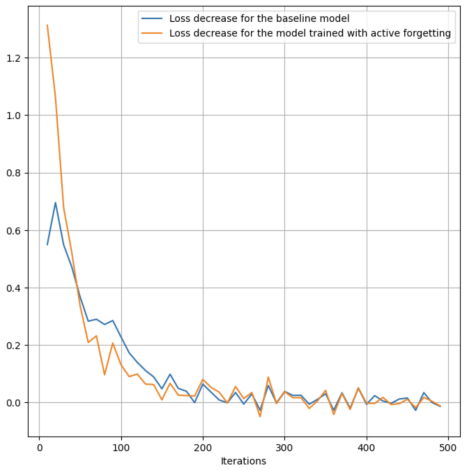

Figure 8: Relative pre-training and fine-tuning loss decrease for standard and Active Forgetting nanoGPT.