UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

Improving Compute-Efficiency of Federated Large Language Model Training with Heterogeneous Clients

Andrzej Szablewski

Churchill College

June 2025

Submitted in partial fulfillment of the requirements for the
Master of Philosophy in Advanced Computer Science

Total page count: 66

Main chapters (excluding front-matter, references and appendix): 53 pages (pp 7–59)

Main chapters word count: 13,926.

Methodology used to generate that word count:

```
texcount report.tex -inc
```

Declaration

I, Andrzej Szablewski of Churchill College, being a candidate for the Master of Philosophy in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this report, I adhered to the Department of Computer Science and Technology AI Policy. I am content for my report to be made available to the students and staff of the University.

Signed: Andrzej Szablewski

Date: 23rd June 2025

Abstract

Federated cross-silo training of large language models enables smaller institutions to collaborate on training foundation models in a decentralised fashion. While it does not require participants to use a single highly-capable data centre, it introduces unique inefficiencies when contributing parties vary in compute capacity and communication bandwidth. This dissertation investigates how to maximise computational efficiency and convergence in such hardware-heterogeneous settings. First, it characterises client idling caused by disparities in mini-batch processing times and mini-batch size limits, proposing four resolution strategies, one of which achieves particularly low idling time without a decrease in model performance. Second, it empirically studies federated hyperparameter optimisation under hardware heterogeneity by measuring gradient noise scale and studying learning rates in both federated and centralised settings. Third, it introduces Selective Multi-Head Attention (SMHA), which recognises redundancy among transformer attention heads and trains only a subset of heads per client, with static and dynamic head ratio schedules driven by neural representation similarity metrics. Finally, thorough experiments on small GPT models show that combining heterogeneity-aware workload balancing with SMHA can significantly shorten wall-time, incurring only a slight decrease in convergence rate, and offering a practical path toward more efficient decentralised LLM pre-training.

Acknowledgements

This project would not have been possible without the incredible support of Lorenzo Sani, who dedicated countless hours introducing me to the challenges of modern federated learning and was always open to discussing my ideas and debugging technical issues. Furthermore, I am grateful to Prof. Nic Lane, who supported me throughout the project and fostered a welcoming atmosphere that made me feel like a true member of the CaMLSys group. Moreover, I would like to thank Alex Iacob for the numerous insightful discussions on neural networks' foundational workings and his invaluable writing suggestions. Finally, I would like to express my deepest gratitude to my parents, whose unwavering encouragement and support have been a constant source of inspiration throughout this project.

In memory of my grandparents, who always supported me in achieving academic success.

Contents

Chapter 1

Introduction

Federated learning (FL) has emerged as a paradigm for training machine learning models across decentralised data sources without requiring raw data exchange (McMahan et al., 2023). At the same time, Large Language Models (LLMs) built on the Transformer architecture (Vaswani et al., 2023) have demonstrated remarkable capabilities when pre-trained on massive text corpora using self-supervised objectives (Brown et al., 2020; Kaplan et al., 2020; Hoffmann et al., 2022). Bringing these two advances together resulted in the possibility of federated cross-silo pre-training of LLMs. This new application of FL is particularly useful for institutions, researchers and general communities that lack single-site computational resources, enabling collaborative training of foundation models via pooled, geographically distributed hardware. However, this setting also has several challenges that, when not addressed, likely decrease the computational efficiency of the large-scale model pre-training.

Firstly, federated training introduces new hyperparameters at both the client and federation levels, including the number of communication rounds, the number of local optimisation steps per round, the mini-batch size on each client, and the choice of inner and outer optimisers. Tuning these federated hyperparameters is critical for convergence speed. Still, it is complicated because clients may vary widely in compute capacity, network bandwidth, and latency, resulting in hardware underutilisation when strict synchronisation is enforced (Yuan et al., 2023).

Secondly, pre-training LLMs at scale demands significant computational budgets. Models with billions of parameters require days or weeks of training, with any inefficiency in local training or communication translating into substantial energy and training wall-time increases (Sani et al., 2024a). In heterogeneous cross-silo settings, where some participants have access to high-end multi-GPU servers and others only own modest clusters, choosing a homogeneous configuration risks slowing down the federation to the speed of the slowest participant, or unevenly allocating workloads that slow down the global convergence.

To address these challenges, in this dissertation I set out three main objectives. Initially,

I aim to characterise and mitigate client idling in heterogeneous FL by analysing how disparities in micro-batch processing times and maximum feasible batch sizes lead to imbalanced workloads, and propose a solution adjusting the number of per-client samples or local iterations. Furthermore, I look into federated hyperparameter optimisation under compute heterogeneity, building on empirical measurements of gradient noise scale and compute-optimal batch sizes. Extending the consideration to model hyperparameters, I observe that attention heads in the Transformer may carry redundant information, while requiring substantial compute. Taking advantage of their similarities, I propose Selective Multi-Head Attention, an architecture-level intervention that trains only a subset of attention heads on each client.

This project produces the following key contributions:

- An analysis of idling time in cross-silo federated training with heterogeneous micro-batch speeds, resulting in multiple hardware-heterogeneity resolution strategies. Notably, the top-performing method results in 8× less per-client idling time.
- Empirical studies of compute-optimal learning rate and gradient noise scale in both centralised and federated settings, suggesting robust hyperparameter selection for heterogeneous clients.
- The proposal of Selective Multi-Head Attention (SMHA), with both static and dynamic head-ratio scheduling, followed by a thorough analysis of how fractional-parameter training influences LLM training dynamics.
- A comprehensive experimental evaluation on small-scale GPT models, demonstrating that the heterogeneity-aware strategies combined with SMHA reduce training wall-time by 14%, but lead to a slightly lower convergence rate.

This dissertation is structured into seven chapters. Following this Introduction, I describe the essential background information, which forms the basis of the problem, proposed solutions, and the experimental studies in Chapter 2. Furthermore, I describe related works in Chapter 3. In Chapter 4, I introduce the problem and describe the methodology, explaining both the hardware-heterogeneous optimisation hyperparameters as well as the dynamic neural architecture intervention. Chapter 5 described the experimental setup, including the details of simulated federation participants as well as the choice of model and training data. Following that, I demonstrate and discuss the results in Chapter 6. Finally, the main body of this dissertation is finalised by Chapter 7, mentioning conclusions, limitations and avenues for future studies. This document ends with two Appendices A and B, the first of which summarises the developed software, while the second includes complementary experimental results.

Chapter 2

Background

2.1 Federated Learning

2.1.1 Setup and Training Objective

Federated learning (FL) is a distributed training paradigm in which a central server and many client devices collaboratively optimise a shared global model without ever exchanging raw data. Each client c holds a local partition of a shared dataset $\mathcal{D}$, where $\mathcal{P}_\rfloor$ are the set of indices associated with client c. Each client c has a partition of n_c samples (x_i, y_i) and a copy of the model parameters w, and defines its own empirical loss:

$$F_c(w) = \frac{1}{n_c} \sum_{i \in \mathcal{P}_\rfloor} l(w; x_i, y_i).$$

The overall objective of federated learning is the minimisation of the weighted sum of the local losses:

$$f(w) = \sum_{c=1}^{C} \frac{n_c}{C} F_c(w).$$

Common aggregation strategy, FedAvg (McMahan et al., 2023), solves this by repeating synchronous communication rounds: at each round t, the server samples a subset of clients, sends them the current global weights $w^{(t)}$, and each client performs a fixed number of local stochastic gradient descent (SGD) steps on its F_c, producing updated weights $w_c^{(t+1)}$. The server then aggregates these updates by weighted averaging of client models to form the new global model $w^{(t+1)}$.

In large model training, simple SGD optimisers are commonly replaced with adaptive methods (e.g. AdamW (Loshchilov and Hutter, 2019)), which are less sensitive to the optimisation hyperparameters. Using different inner and outer optimiser, results in a general bi-level optimisation framework, in which clients supply the server with updated per-round model parameters trained with the inner optimiser, further used by the server

to compute the pseudogradient $\Delta^{(t+1)} = \frac{1}{|C|}\sum_{c\in C}(w^t - w_c^{(t+1)})$. The pseudogradients are used by the outer, federation-level optimiser. Such a general framework allows for more advanced optimisation methods, such as aggregating optimiser states (Cheng and Glasgow, 2025).

2.1.2 Federated Hyperparameters

The decentralised federated setting results in the emergence of additional hyperparameters, influencing the optimisation. A key federated hyperparameter is the number of communication rounds R. This determines how many times clients synchronise with the server: increasing R generally improves model convergence but also raises the total communication cost and latency before deployment. Assuming constant bandwidth between clients, growing model size leads to longer synchronisations. If synchronisation periods are very long, they may lead to significant bottlenecks and hardware underutilisation. A closely related hyperparameter is the client participation fraction – the proportion of all clients selected each round. While it is essential in the context of non-IID data partitioning, this work focuses on scenarios where clients have access to the full shared dataset and data is IID partitioned.

Another hyperparameter is the number of local steps L, which translates to the number of mini-batch gradient updates each client performs before returning its model to the server. If each client's dataset has n_c examples and the batch size is B, then one full epoch corresponds to $L = \lceil n_c/B \rceil$ steps. Under IID data, increasing L straightforwardly reduces the total number of communication rounds needed for convergence, since each client's update becomes a more accurate estimate of the global gradient. In practice, however, very large L can yield diminishing returns and be limited by the high costs of frequent communication.

The mini-batch size controls the granularity of each gradient estimate. A larger B reduces the variance of the stochastic gradient, which can permit a larger learning rate and faster per-step convergence. Conversely, a smaller B injects more noise into each update, which, under IID data, can help explore the loss landscape and improve generalisation. Notably, the mini-batch size is strictly related to the number of local steps L, under a constant training budget T.

The learning rate must be tuned to accommodate L and B. Under IID data, the standard convergence analyses for SGD apply: if the learning rate is too large, updates will oscillate or diverge; if too small, progress per round will be slow, and many more communication rounds will be needed. A common strategy is to scale the learning rate roughly linearly with B (so that the expected per-step variance remains constant) and to decay it over the course of training with a learning rate scheduler. The challenge of learning rate scaling can also be approached using square root scaling or other methods (Li et al., 2024).

Hyperparameter	Symbol	Description
Number of communication rounds	R	Number of times clients synchronise with the server. Increasing R generally improves convergence but increases total communication cost and latency.
Client participation fraction	P	Proportion of all clients selected each round. Affects convergence speed and variance, particularly under non-IID data. In this work, I consider cross-silo setting and assume $P = 1$.
Number of local steps	L	Number of mini-batch gradient updates each client performs before aggregation. Larger L reduces needed communication rounds but increases local compute.
Mini-batch size	B	Number of examples per local gradient update. Larger B lowers gradient variance (allowing larger learning rates), while smaller B adds noise that can aid exploration.
Learning rate	η	Step size for each gradient update. Usually tuned along B and L: too large causes divergence; too small slows convergence. Often scaled with B and decayed over time.

Table 2.1: Key hyperparameters in a decentralised federated learning setting.

2.1.3 Cross-Silo Setting

Cross-silo federated learning is a collaborative training paradigm in which a small, fixed set of organisationally distinct “silos" (e.g. banks, hospitals, scientific institutions) jointly train a shared global model by exchanging only model updates rather than raw data. Recently, this method has been proved successful in decreasing training times of particularly large models (Sani et al., 2024a; Douillard et al., 2025), conceptually acting as a single decentralised data centre. Cross-silo FL rests on several foundational assumptions that differentiate it from cross-device scenarios. First, the system comprises a small number of participants, whose membership remains stable throughout training. Each silo is expected to be reliably online and responsive in every communication round. Because of significant compute resources, silo-sized clients can perform multiple local iterations with large batch sizes. However, only some institutions are equipped with high-bandwidth network links between data centres, which allow the exchange of billion-parameter models, while many others use commercial network links. In those cases, a large number of communication rounds results in client idling and a significant amount of time spent exchanging parameters. To avoid the severe impact of this issue in a large model regime, the number of communication rounds needs to be carefully chosen.

2.2 Large Language Models

Federated learning introduces additional hyperparameters, while infrequent parameter synchronisations effectively modify the optimisation procedure. Other key factors in training a Large Language Model are the hyperparameters defining its neural architecture. Hence, a comprehensive understanding of the field of LLMs is important.

Large Language Models are deep neural networks, generally based on the Transformer architecture, and pre-trained on massive text corpora via self-supervised objectives such as next token prediction (causal language modelling). By scaling both the number of parameters and the diversity of training data, LLMs learn rich, contextual representations that enable them to generate human-like text. Pre-training typically leverages vast distributed compute to optimise the autoregressive objective, after which models can be adapted through fine-tuning. This work focuses on the former.

2.2.1 Generative Pre-Trained Transformer Architecture

To understand the architectural changes described further in this work, it is essential to familiarise oneself with key components of Generative Pre-Trained Transformer (GPT) (Radford et al.). This neural network consists of l identical Transformer decoder blocks, each of which transforms an input token embedding into a contextualised representation through two main components: an attention layer and a feedforward layer. The former updates hidden representations, respecting the patterns present in the sequence of input tokens, while the latter applies non-linear transformations to the updated hidden representations. The rest of this study focuses on the attention mechanism.

2.2.2 Multi-Head Attention

Attention is a commonly used function in neural architectures to transform between discrete input sequences. It can be intuitively understood as a transformation of each sequence element into an average of other input elements, weighted by the pairwise similarity between each pair.

To compute attention in practice, each d_{model}-dimensional sequence element is linearly projected into three l-dimensional representations: query, key and value vectors using corresponding weight parameters. Assuming an n-dimensional input sequence, we obtain:

$$\mathbf{Q} = \mathbf{X}\mathbf{W_q} \quad \mathbf{K} = \mathbf{X}\mathbf{W_k} \quad \mathbf{V} = \mathbf{X}\mathbf{W_v},$$

where $\mathbf{X} \in \mathbb{R}^{n \times d_{model}}$ represents the input sequence, and $\mathbf{W_q}, \mathbf{W_k} \in \mathbb{R}^{d_{model} \times d_k}$ and $\mathbf{W_v} \in \mathbb{R}^{d_{model} \times d_v}$ are the projections. Practically, d_k and d_v are usually set equal.

The original implementation of attention in the Transformer, the Multi-Head Attention,

computes attention multiple times, over distinct sets of weights. The outputs of those so-called heads are subsequently aggregated using another learnt linear projection. The attention output for the ith head, is given by:

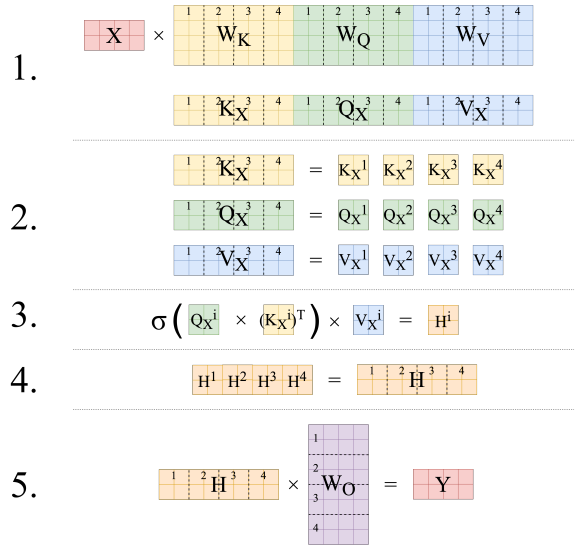
$$\mathbf{H^{(i)}} = \text{softmax}(\frac{\mathbf{XW_Q^{(i)}}(\mathbf{XW_K^{(i)}})^T}{\sqrt{d_k}})\mathbf{XW_V^{(i)}},$$

with $\mathbf{H^{(i)}} \in \mathbb{R}^{n \times d_v}$. The Multi-Head Attention (MHA) with n_{head} heads is obtained by subsequently:

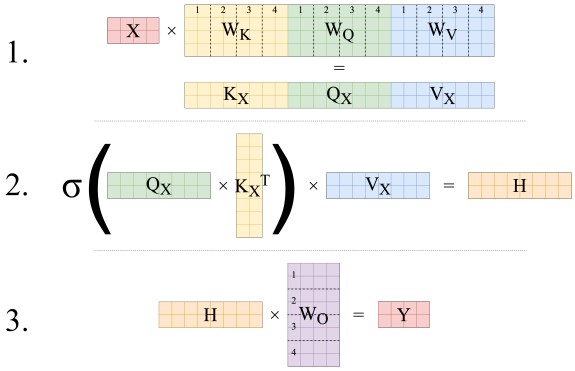
$$\mathbf{H_{concat}} = \text{Concat}_2([\mathbf{H_1}, \mathbf{H_2}, ..., \mathbf{H_n}])$$

$$\text{MHA}(\mathbf{X}) = \mathbf{H_{concat}}\mathbf{W_o},$$

where $\mathbf{W_o} \in \mathbb{R}^{n_{head}d_v \times d_{model}}$ and Concat_2 joins matrices in the second dimension.



(a) In the original MHA implementation, keys, queries and values are computed simultaneously for all attention heads by a joint parameter matrix W_{KQV}.



(b) To accelerate training, computations of attention weights can be batched together, eliminating step 2. and combining steps 3. and 4. Note, that this results in replacing $2m$ multiplications with just two matmul operations.

Figure 2.1: Multi-Head Attention component within a Transformer layer. Note that, because of efficiency reasons, MHA batches and parallelises the computation of all heads. The attention scaling factor is omitted in both diagrams for brevity.

Chapter 3

Related works

3.1 Federated Learning

3.1.1 Decentralised Large Model Training

Cross-silo federated learning has recently been applied to training very large neural architectures because of its advantage to decentralise training on expensive computing hardware across multiple participants (Yuan et al., 2023; Sani et al., 2024a,b; Douillard et al., 2024). Photon, the first end-to-end system for federated LLM pre-training, demonstrates training a 7B parameter model, exceeding centralised perplexities (Sani et al., 2024a). Furthermore, Sani et al. (2024a) show it is possible to reduce communication by up to 3 orders of magnitude through infrequent cross-silo synchronisation. Building on this, Iacob et al. (2024) propose WorldLM, a hierarchical “federation of federations" framework that partitions models into a shared backbone and personalised layers. OpenFedLLM integrates seven federated algorithms to fine-tune Llama2-7B on private user data for downstream instruction tuning and value alignment, demonstrating superior performance over local baselines (Ye et al., 2024). To further decrease bandwidth, DiLoCo employs local AdamW updates with momentum-equipped outer optimiser, and hundreds of local steps per synchronisation, achieving parity with synchronous training at 500$\times$ lower communication (Douillard et al., 2024). Finally, Streaming DiLoCo overlaps sequential synchronisation with computation to significantly reduce peak bandwidth and training wall-time (Douillard et al., 2025). To validate the empirically observed benefits of decentralised, cross-silo large model training, He et al. (2024) show that local SGD obeys power-law scaling of validation loss with model size, matching distributed data-parallel performance.

3.1.2 Heterogeneous Hardware FL

Federated learning must contend with client hardware heterogeneity. Client devices differ in compute speed, memory, and network link, which may lead to slower delaying synchro-

nised training rounds, or worsening the overall convergence. Algorithmic approaches may adapt the FL optimisation itself to account for straggling clients. For example, FedProx introduced flexible local update regimes, allowing faster clients to perform more epochs of training while limiting slower ones, accounting for the inconsistent local workloads (Li et al., 2020). Similarly, FedNova uses a normalised averaging technique to correct for different amounts of work done per client, eliminating the bias in standard FedAvg when clients perform heterogeneous numbers of local steps (Wang et al., 2020).

Another line of work is to relax the strict synchronisation of FedAvg via asynchronous federated learning, such as in the ASO-FL framework introduced by Chen et al. (2020) or recently DiLoCo by Douillard et al. (2024). In asynchronous FL, the server does not wait for all clients in a round, and instead it updates the global model whenever a client's result arrives. However, the challenge becomes dealing with stale updates: theoretical convergence analyses show that the convergence rate of asynchronous FL often depends on the maximum delay of client updates.

Finally, beyond client selection, researchers have looked at resource-aware local training hyperparameter optimisation. For example, Sani et al. (2024a) demonstrate the use of a significantly larger learning rate on the clients, which would lead to model divergence in a centralised setting.

3.2 Dynamic & Sparse Transformer Architectures

Modern approaches to dynamic and sparse neural architectures seek to adaptively prune or route computation to match available resources and input complexity. However, this is not directly feasible in dense architectures like the original Transformer, since output computation depends on all model parameters. Hence, several methods for sparsifying the transformer have been proposed.

One of the methods is Mixture of Experts, employed in a Switch Transformer, which trains multiple feedforward components per layer, but uses only some of them for next token prediction (Fedus et al., 2022). Similarly, SwitchHead introduces MoE to attention layers by maintaining a small number of attention matrices (heads) but equipping each with multiple value/output experts (Csordás et al., 2024b). Complementing this, the Mixture of Attention Heads (MoA) mechanism routes each token to a learned subset of k attention-head experts. Following that, a lightweight router network selects, for each token, the most relevant heads from a larger pool, enabling the model to scale the number of active heads dynamically per input while maintaining expressivity (Zhang et al., 2022). Finally, MoEUT uses fine-grained MoE blocks in feedforward and self-attention layers, exceeding standard Transformer performance in language modelling with far fewer compute and memory requirements (Csordás et al., 2024a). Crucially, Xia et al. (2023) studied how increasing model capacity by resizing parameter matrices in a Vision Transformer modifies

model behaviour and maximises the gains from a limited training budget.

Sparsity has also been explored in Federated learning through methods such as SparsyFed, which dynamically computes sparse gradients for only some parameters, saving local compute (Guastella et al., 2025). Other sparsity-enhancing methods were developed to improve communication efficiency or the objective convergence in non-IID settings (Hu et al., 2022). However, note that a limited number of works focus on reducing local client computation.

Chapter 4

Methodology

This chapter starts with an overview of the core problem and the outline of the federated setting, presented in Section 4.1. It continues in Section 4.2 with the analysis of optimisation hyperparameters, which are crucial for maximising compute efficiency of federated learning (FL), especially when clients differ in their hardware capabilities. The chapter continues by proposing a modification to the neural architecture of Transformer models – Selective Multi-Head Attention (SMHA), and describing its thorough study in the centralised setting (Section 4.3). Finally, it describes the proposed combination of the compute-efficiency improving strategies with the application of SMHA to hardware-heterogeneous FL in Section 4.3.4.

4.1 Problem Statement

This dissertation focuses on a particular use case of federated learning, intended as collaborative cross-silo large model pre-training. The advantages of this method over the single-site training include a shorter training wall-time, the possibility to aggregate computing power from multiple sparsely-connected sites, and potential access to higher-quality training data. These benefits can be crucial enabling factors for smaller institutions, such as universities or research labs, to collaborate on training models with capabilities they could not achieve alone. Notably, the federated learning setting this dissertation is interested in can be characterised by the following properties:

- IID Dataset Partitioning. The clients source training samples from a federated dataset, a large text corpus sharded across them. Each shard has independently and identically distributed samples. Note that in the case of LLMs, the samples in pre-training corpora are often assumed to be IID. Hence, this work may also be relevant for scenarios when clients use private data.
- High-Performance Clients. The participants of the collaborative training are sufficiently funded to afford the use of multi-GPU hardware.

- Sparse Synchronisation. Frequent synchronisation of the model parameters, such as once per mini-batch as in the distributed data parallel pipelines, is impossible due to the latency and bandwidth constraints between the clients and between the clients and the server.
- Dense Neural Architecture. The neural model is built on a dense architecture, so that each client must be able to hold a full copy of the model at all times. Hence, it is not designed to be trained separately.

In federated environments with no or limited system heterogeneity among clients, the participants usually share a single configuration of local training hyperparameters. This, in turn, results in similar per-round contributions from every client, as well as performing the same number of local optimisation steps, each taking almost identical duration. Given similar network bandwidth between clients, the federation can synchronise immediately after each round, with minimal client idling time. A similar level of client contributions simplifies tuning the federation hyperparameters, such as the global learning rate.

Other federated environments facilitate the expansion of the number of contributors by admitting heterogeneous computing devices among clients and communication over heterogeneous networks. For example, smaller institutions involved in a collaborative pre-training of foundation models often differ in access to computing hardware. Hence, different hardware capabilities of the clients pose an additional challenge of optimising each of them separately. To minimise the underutilisation of the system caused by the difference in processing times of faster and slower clients, it is possible to modify the number of local steps and the amount of data processed by each of them. However, when clients apply different hyperparameters for their local optimisation, their per-round contributions will likely differ in quality. This, in turn, may lead to sub-optimal convergence rate and final performance of the aggregate model. Participating sites may also differ in access to stable high-bandwidth network links, causing additional delays. However, this dissertation does not focus on the problem of communication, assuming a stable, intra-region connection between computing sites.

Summarising, this work investigates the choice of hyperparameters in FL with heterogeneous hardware, optimising for two objectives:

1. **Minimising the idling time of the computing systems**,
2. and **achieving the lowest validation loss**.

4.2 Federated Learning with Heterogeneous Hardware

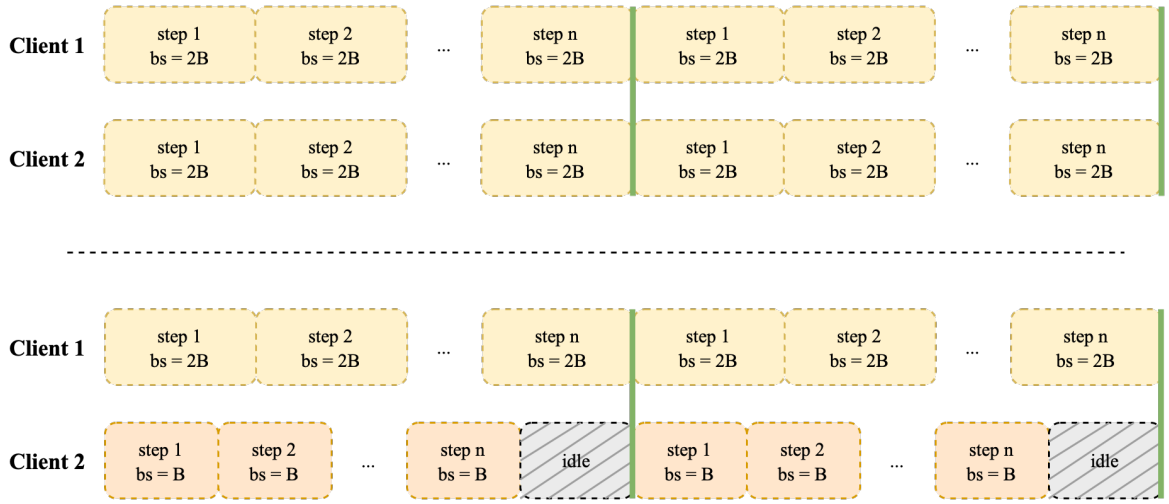


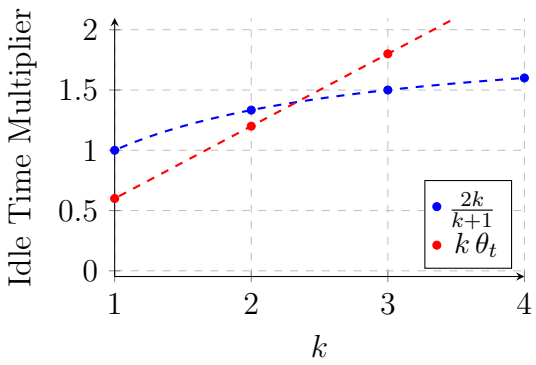
Figure 4.1: Differing client hardware influences the duration and the number of samples of a local step. This, in turn, results in idling of one of the clients before the synchronisation point (green vertical line). **Top**: two clients with identical compute capabilities. **Bottom**: client 2 with less capable hardware than client 1.

In a standard, hardware-homogeneous setting, clients use identical hardware. Hence, the hardware-related aspects of local optimisation are usually identical as well. This means that the maximum micro-batch size B_{max} (a maximum number of samples that fits into accelerator memory), as well as the corresponding processing time of a micro-batch t_{bmax}, are the same across participants. In the heterogeneous case, hardware differences directly influence the maximum micro-batch size and processing time, leading to distinct values for each of the clients. To understand how this differs from the case of identical client compute capabilities, let's consider the following toy example and assume a fixed training budget of T samples, fixed number of server rounds R, but two different clients C_1 ($B_{max} = B_1$, $t_{bmax} = t_1$) and C_2 ($B_{max} = B_2$, $t_{bmax} = t_2$), where $B_1 = kB_2$ for some k and $t_1 > t_2$. Proceeding similarly to the homogeneous case results in client C_2 processing $k\times$ fewer samples than C_1, as shown in Figure 4.1. This is an issue, since the full training budget T cannot be achieved without increasing the number of rounds R. I explore several **strategies** to proceed about this heterogeneity:

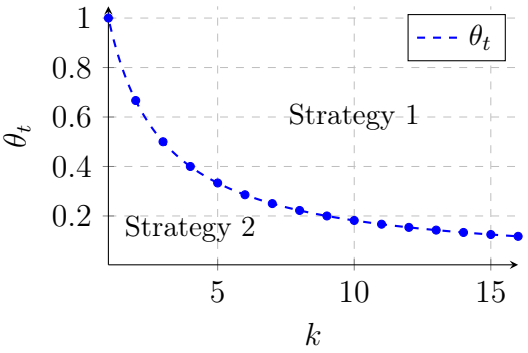
1. Proceed with the same number of iterations per client within a round. Apart from different numbers of samples processed by each of the clients per round, the number of local steps needs to be rescaled by $\frac{2B_1}{B_1+B_2}$ to use the entire training budget T (Figure 4.3a).

2. To stay consistent with the number of samples processed by each client per round, client C_2 performs $k\times$ the number of C_1's micro-batches (Figure 4.3b). Two cases arise:

   (a) Both clients set their mini-batch sizes equal (C_2 uses gradient accumulation), allowing them to use the same learning rate.



(a) Comparison of the per-round idle-time overhead of Strategy 1 and both instances of Strategy 2, for an example value of $\theta_t = 0.6$, empirically measured using NVIDIA A40 GPU in small LLM training. Note that k is a multiple of a mini-batch size, and hence the domain is integral. For smaller k, Strategies 2a and 2b will result in a shorter idle time. For larger k, Strategy 1 is preferable for minimising the idle time.



(b) The boundary mini-batch process time ratio θ_t between which strategy minimises the idle-time, for all values of mini-batch size multiples k. Given the asymptotic shape of the curve, growing k initially requires a significant decrease in θ_t for Strategy 2 to be optimal. The decrease further plateaus. Note that k can be integral only.

Figure 4.2: Formal analysis of minimal idle-time between Strategy 1 and both instances of Strategy 2.

(b) Client C_2 performs $k\times$ more optimiser steps with a lower learning rate, reflecting the centralised theory.

Compared to a homogeneous case, where both clients have C_1's specification, the first scenario extends the training duration by a factor of $\frac{2B_1}{B_1+B_2} = \frac{2k}{k+1}$, during which C_2 is idling. On the other hand, the next two approaches leave the number of local iterations unchanged, but extend the round's duration by approximately $\frac{kt_2}{t_1}$, during which C_1 is idling.

Note that the idling time in both cases, 2a. and 2b., scales linearly with k with a constant $\frac{t_2}{t_1}$. On the other hand, in the first case, the time multiplicative factor $\frac{2B_1}{B_1+B_2} = \frac{2k}{k+1}$ is asymptotically bound by:

$$\lim_{k\to\infty} \frac{2k}{k+1} = 2.$$

Hence, to study which of the first three scenarios results in the shortest idling time of the system, it is necessary to empirically measure the ratio of micro-batch processing times $\theta_t = \frac{t_2}{t_1}$. Figure 4.2a presents the idle time scaling for $\theta_t = 0.6$. Solving $\frac{2k}{k+1} = k\theta_t$ for θ_t, results in two regimes:

- $\theta_t < \frac{2}{k+1}$ results in a shorter idle time in strategies 2a. and 2b.
- $\theta_t > \frac{2}{k+1}$ results in a shorter idle time for strategy 1.

Figure 4.2b shows the boundary function $\theta_t(k) = \frac{2}{k+1}$ between the two regimes.

4.2.1 Minimising Client Idle Time

To limit the idling of C_1 or C_2, both clients can perform a different number of mini-batches per round to minimize the time difference arising between them[1]. This, however, may result in a different number of samples processed by each client. I refer to this strategy as strategy 3, and present in Figure 4.3c.

To estimate the duration and the amount of idling of this last approach, it is necessary to formalise this problem further. To minimise the idling of the clients, I aim to find $n, m \in \mathbb{Z}_+$, such that:

$$\min_{n,m} \left| nt_1 - mt_2 \right|, \tag{4.1}$$

$$\text{subject to} \tag{4.2}$$

$$S = nB_1 + mB_2, \tag{4.3}$$

where $S = \frac{T}{R}$ is the total number of samples per round across clients. The aim of the constraint is to ensure the total number of samples processed in a round is S. Rewriting the constraint to $S = (nk+m)B_2$, simplifies the problem, which can be solved by substituting and rearranging the terms, leading to the real-valued minimiser $n^* = \frac{\frac{S}{B_2}t_2}{t_1+kt_2}$. Given n needs to be integral, the solution is either $\lfloor n^* \rfloor$ or $\lceil n^* \rceil$, depending on which pair (n, m) minimises the problem. Finally, the duration of the round is given by nt_1 or mt_2, whichever is larger, while the idling time is given by their difference.

Generalising to C Clients

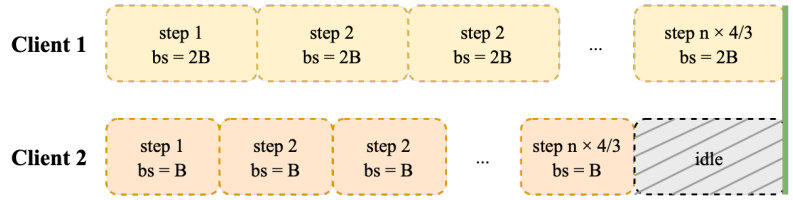
Minimising the client idle time becomes even more important when the number of participating clients increases, because even a single less capable client can result in a significant underutilisation of the federation. This problem can be generalised to arbitrary S, and C clients, each with a different pair of configuration ($B_{max} = B_i$, $t_{bmax} = t_i$). While minimising the sum of all client idle times $\sum_{i=1}^{C} (\min_j (n_j t_j) - n_i t_i)$, would lower the total idle time in the system, the clients are not identical, and the cost of their underutilisation is not the same. Hence, similarly to the case $C = 2$, I decide to minimise the maximum pairwise difference between processing times for any two clients. For $n_i \in \mathbb{Z}_+$, the problem becomes:

$$\min_{n_i} \max_{i<j} \left| \, n_i \, t_i - n_j \, t_j \right|, \tag{4.4}$$

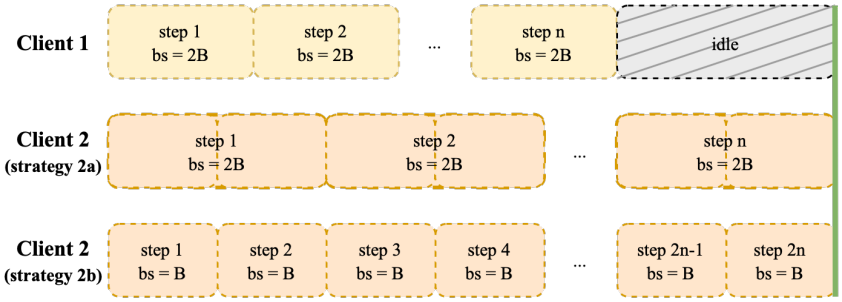
$$\text{subject to} \tag{4.5}$$

$$S = \sum_{i=1}^{C} n_i \, B_i. \tag{4.6}$$

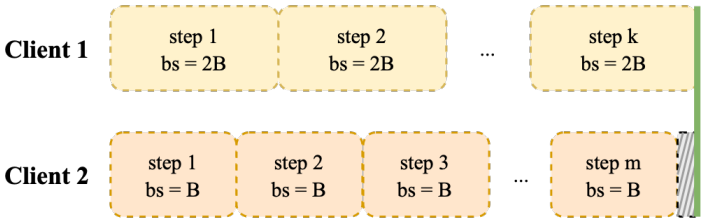
[1]Clients may perform more than one micro-batch per optimiser iteration, but for the simplicity of this example, I assume clients avoid accumulating gradients, if possible.



(a) Strategy 1.



(b) Strategies 2a and 2b.



(c) Strategy 3.

Figure 4.3: Hardware-heterogeneity resolution strategies.

This revised task is an instance of a minimax problem, involving minimising the largest difference between any two products $n_i t_i$. Conveniently, the minimax objective can be turned into a linear objective by introducing auxiliary variables $M = \max_i n_i t_i$, $m = \min_j n_j t_j$, converting

$$\min_{n_i} \max_{i<j} \left| n_i\, t_i - n_j\, t_j \right| = \min_i \left[\max_i n_i t_i - \min_j n_j t_j \right] \tag{4.7}$$

into:

$$\min\,(M - m) \tag{4.8}$$

$$\text{subject to} \tag{4.9}$$

$$n_i t_i \leq M, \quad n_i t_i \geq m. \tag{4.10}$$

$$S = \sum_{i=1}^{C} n_i\, B_i. \tag{4.11}$$

This is a mixed-integer linear program, which we efficiently solve for a relatively small number of clients $C < 100$ using a MILP solver by enforcing bounds on the problem variables. Once solved, the round duration is given by $\max_i t_i$, while the amount of idling can be inferred as $\sum_{i \neq j}^{c} \left[\arg\max_j (t_j) - t_i\right]$.

Summary of the Heterogeneity Resolution Strategies

To summarise the two-client scenario, the following Table 4.1 presents the homogeneous ($B_1 = B_2$, $t_1 = t_2$) case, as well as the four considered strategies of handling hardware heterogeneity of 2 clients C_1, C_2.

	Steps/round		**Samples/round**		**Round time**	
Strategy	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_1}$	$\mathbf{C_2}$
homo.	L		$L \times B_1$		$L \times t_1$	
1.	$L \times \frac{2B_1}{B_1+B_2}$		$L \times \frac{2B_1^2}{B_1+B_2}$	$L \times \frac{2B_1B_2}{B_1+B_2}$	$L \times \frac{2B_1}{B_1+B_2} \times t_1$	$L \times \frac{2B_1}{B_1+B_2} \times t_2$
2a.	L		$L \times B_1$		$L \times t_1$	$L \times k \times t_2$
2b.	L	$L \times k$	$L \times B_1$		$L \times t_1$	$L \times k \times t_2$
3.	n	m	$n \times B_1$	$m \times B_2$	$n \times t_1$	$m \times t_2$

Table 4.1: Summary of how the key federated hyperparameters change depending on the choice of the strategy. Top row shows a comparison to a homogeneous-hardware baseline with two clients C_2.

4.2.2 Choice of the Mini-Batch Size

The LLM pre-training generally falls into the large mini-batch-size regime, as described by (McCandlish et al., 2018). Its authors empirically show that when training large models in a highly data-parallel environment, there exists a compute-optimal critical mini-batch size B_{crit}. Increasing the mini-batch size until B_{crit} increases the rate of convergence of the machine learning objective. However, growing mini-batch size past B_{crit} results in diminishing returns. To chose an optimal combined mini-batch size B_{eff} for the hardware-heterogeneous clients, I follow the methodology described in the paper: find an optimal learning rate, measure the gradient noise scale $\mathcal{B}_{noise}$ at each step, and set B_{crit} to the average of $\mathcal{B}_{noise}$ of the training. We approximate the $\mathcal{B}_{noise}$ as $\mathcal{B}_{simple} = \frac{\mathrm{tr}(\Sigma)}{|G|^2}$, where Σ is the covariance of the gradient of the optimisation step G. To get an unbiased estimate of $\mathcal{B}_{simple}$, I use the methodology described in Appendix A.1 of the work by McCandlish et al. (2018).

Learning Rate

To choose an optimal learning rate, which is necessary for reliable B_{crit} estimation, I search for the learning rate in a pre-defined interval, with a reasonable mini-batch size value. To do so, I use a centralised setting, and apply the identified learning rate value to the federated case. However, the chosen mini-batch size is going to affect how reliable the estimation of the learning rate value is. Hence, I study whether the centralised learning rate search should be based on the sum of the mini-batch sizes of each clients (the effective mini-batch size B_{eff}), or the average mini-batch size of a single client. To do so, for each studied learning rate value, I compare the convergence of the ML objective in a hardware-

homogeneous federated learning case, where every client uses a mini-batch size of $\frac{B_{eff}}{C}$ to a:

- centralised setting with a mini-batch size B_{eff}, and
- centralised setting with the average mini-batch size of a client $\frac{B_{eff}}{C}$.

Following the study, I choose the centralised setting learning rate that is closer to the hardware-homogeneous FL case. For the heterogeneous mini-batch size across clients, I apply the linear learning rate scaling.

Gradient Noise Scale in the Federated Scenario

While McCandlish et al. (2018) experimented with measuring gradient noise scale in various machine learning scenarios, the metric has not been studied in the context of federated learning. Hence, to understand whether it is even suitable in the setting with infrequent parameter synchronisations, I compare the behaviour of $\mathcal{B}_{simple}$ in two cases:

- Centralised case, with a fixed learning rate (estimated through the procedure described in the above sub-section) and the mini-batch size of B_{eff}.
- Hardware-homogeneous case, with the same learning rate value and the per-client batch size $\frac{B_{eff}}{C}$.

Point readouts of the $\mathcal{B}_{simple}$ are noisy on their own, hence a smoothing described in the original paper is applied. This method only works when $\text{tr}(\Sigma)$ and $|G|^2$ are measured frequently, i.e. every local optimisation step. Measuring the values of per-client $\mathcal{B}_{simple}$ between rounds is not feasible due to a limited number of values.

Note, that the original study does not consider the case when the learning rate changes throughout training. LLM pre-training commonly involves using learning rate decay and warm-up phases, which additionally alter the value of $\mathcal{B}_{simple}$. Hence, I additionally study the mean values in different phases.

Mini-Batch Size and Local Steps

In the case of homogeneous hardware of the clients, the number of local optimisation steps L is tuned together with the mini-batch size and generally set equal for all of the clients. However, when the clients differ in their compute capabilities, it becomes a new, client-specific hyperparameter arising in the setting. To study its influence on the machine learning objective convergence, I combine the strategies introduced earlier in this section with the analysis of optimal mini-batch sizes. Given a fixed number of samples per round, there is a trade-off between the mini-batch size and the number of local steps. Whenever possible, I aim to use the theory-predicted B_{crit}. Otherwise, the available mini-batch size for every client is determined by its B_{max} and the heterogeneity resolution strategy selected.

4.2.3 Evaluation Metrics

To evaluate the quality of the selected hyperparameters, several metrics are used. I measure the model performance by computing the per-round and final validation set perplexity using the aggregated model. To understand the utilisation of the accelerators in the system, I track the per-client and per-round training time, client throughputs, and the total training wall-time.

Given differently configured clients, it is interesting to study how much each of the corresponding local versions of the model changes after a single round. To understand the per-round source of improvement in the aggregated model, I additionally measure validation set perplexity on each instance of a model before (aggregated model from the previous round) and after local optimisation. The corresponding $PPL_{pre}^{(i)}$ and $PPL_{post}^{(i)}$ values can be further compared and studied over time, for every round i. Note, that when using FedAvg, I expect the $PPL_{pre}^{(i+1)}$ to be lower than $PPL_{post}^{(i)}$, due to the aggregation and averaging of individual pseudogradients. Hence, given a client c and a round i, I define a **client gap ratio** as:

$$G_c^{(i)} = \frac{PPL_{post}^{(i)}(c)}{PPL_{pre}^{(i+1)}(c)}.$$

4.3 Selective Multi-Head Attention

So far, this study has focused on the hyperparameters related to the federated setting and the process of optimisation. To further compensate for the varying hardware capabilities of the federated clients, one can look for possible changes in the neural model architecture. However, training local models with arbitrarily varying neural architectures requires one to resolve how to aggregate parameters of models with possibly varying numbers of layers, or parameters within a layer.

The neural architecture typically used for LLM training at scale – the Generative Pre-Trained Transformer (GPT) – has been extensively studied in the direction of the usefulness of its key components – the attention and feedforward layers. Specifically, there have been growing efforts towards understanding their role in generating human-like text. However, many of their aspects are still unexplored. In the effort to further equalise the computation of heterogeneous federated clients, we focus on one of the aspects in particular: the compute-optimal number of attention heads in each transformer block.

The use of the originally proposed Multi-Head Attention mechanism or its generalisation (Group-Query Attention) by design allows for independently computing all attention components (queries, keys, values) for each head. Hence, it is possible to compute only a subset of them at each iteration. Recent studies suggest that different attention heads learn similar transformations and produce similar attention maps, indicating some level of redundancy (Jo and Myaeng, 2020). From the computational perspective, the redun-

dancy is not desired, since it unnecessarily consumes resources. Building on top of these findings, the following research question arises:

Can different federated clients share a copy of the same model but train only some of its attention heads?

4.3.1 SMHA Design

Following Xia et al. (2023), who considered a similar concept in the context of computer vision, we reformulate the attention mechanism introduced in Section 2.2.2 as follows, to highlight the independent computation for each head:

$$\mathbf{H^{(i)}} = \text{softmax}(\frac{\mathbf{X}\mathbf{W_Q^{(i)}}(\mathbf{X}\mathbf{W_K^{(i)}})^T}{\sqrt{d_k}})\mathbf{W_V^{(i)}}. \tag{4.12}$$

The final projection can be expressed as a concatenation of block matrices

$$\mathbf{W_o} = \text{Concat}_1([\mathbf{W_o^{(1)}}, \mathbf{W_o^{(2)}}, ..., \mathbf{W_o^{(n_{head})}}]), \tag{4.13}$$

where $\mathbf{W_o^{(i)}}$ is a $\mathbb{R}^{\frac{d_v}{n_{head}} \times d_{model}}$ projection, corresponding to the output of the ith head. Then, we define Selective Multi-Head Attention (SMHA) over a set of heads I_h as:

$$\text{SMHA}^{I_h}(\mathbf{X}) = \sum_{i \in I_h} \mathbf{H^{(i)}}\mathbf{W_o^{(i)}}. \tag{4.14}$$

This allows to control the number of attention heads in each layer, and therefore to modify the neural architecture. The fraction of the active attention heads over all available heads in each layer l, is defined as **attention ratio** of layer l: $\alpha_{attn}^{(l)} = \frac{|I_h|}{n_{head}}$. Note that the original Multi-Head Attention is the special case of the above formulation, and is achieved with $\alpha_{attn}^{(l)} = 1$, when $|I_h| = n_{head}$ in all layers.

Dynamically changing the number of trained attention heads may have serious implications for the training stability and the end model performance. Hence, I first study its effects in the simple scenario of centralised training. Initially, I compare model training runs varying only in the number of heads trained. Incorporating the results, I further explore whether it is possible to dynamically add attention heads to increase the capacity of the model to capture more sophisticated language patterns.

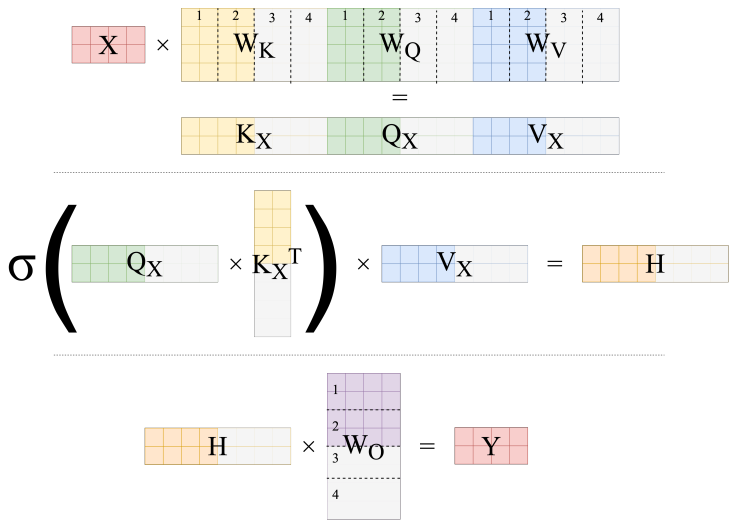


Figure 4.4: Selective Multi-Head Attention component within a transformer layer. Due to the block matrix multiplications used to accelerate training, its sufficient to mask and rearrange attention weights to select the desired heads. The attention scaling factor is omitted for brevity.

4.3.2 Model Comparison Metrics

Language Modelling Performance

To compare different instances of trained language models, I use Perplexity (PPL) as a primary metric. Perplexity is a direct measure of the language modelling objective and is relatively cheap to compute, hence it is reported frequently. Using popular benchmarks of downstream model abilities such as general knowledge (e.g. MMLU), commonsense natural language inference (e.g. HellaSwag) or commonsense reasoning (e.g. Winogrande) has a very limited use-case for small models due to their relatively high difficulty. Using those benchmarks showed unreliability and often resulted in worse than random performance in initial experiments, hence I discontinued using them.

Attention Map Similarity Measure

The key foundation of the idea of pre-training with a deficient number of heads is the redundancy in patterns they capture at each layer. Hence, I need a method to capture and quantify the amount of similarity between contributions of attention heads.

One way to study the attention mechanism is by investigating the so-called attention maps. To compare the behaviour of the distinct attention heads, I compare the maps they produce over identical input token sequences. To quantify the similarity between them, I use Centered Kernel Alignment (CKA) with a linear kernel $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$. Acting as a similarity measure invariant to isotropic scaling and orthonormal transformations, it measures the correlation between the column-spaces of the attention maps and better exposes the syntactic and semantic dependencies between tokens. CKA is defined as:

$$\text{CKA}(\mathbf{K}, \mathbf{L}) = \frac{\text{HSIC}(\mathbf{K}, \mathbf{L})}{\sqrt{\text{HSIC}(\mathbf{K}, \mathbf{K})\text{HSIC}(\mathbf{L}, \mathbf{L})}},$$

where $\text{HSIC}(\mathbf{K}, \mathbf{L})$ is a Hilbert-Schmidt Independence Criterion between input matrices $\mathbf{K}$, $\mathbf{L}$, which can be empirically estimated as follows:

$$\text{HSIC}_b(\mathbf{K}, \mathbf{L}) = \frac{1}{(n-1)^2}\text{tr}(\mathbf{KHLH}),$$

where n is the sample size, and $\mathbf{H}$ is a centering matrix, defined as $\mathbf{H} = \mathbf{I}_n - \mathbf{1}\mathbf{1}^T$.

To get a reliable measure of attention head similarity, I compute HSIC over a large number of input sequences. To correct for the bias of the above estimator, I implement the unbiased version, given by Song et al. (2007) as follows:

$$\text{HSIC}_u(\mathbf{K}, \mathbf{L}) = \frac{1}{n(n-3)}\left[\text{tr}(\mathbf{KL}) + \frac{\mathbf{1}^T\mathbf{K}\,\mathbf{1}\,\mathbf{1}^T\mathbf{L}\,\mathbf{1}}{(m-1)(m-2)} - \frac{2}{m-2}\mathbf{1}^T\mathbf{KL}\,\mathbf{1}\right],$$

where $\bar{\mathbf{K}}$ and $\bar{\mathbf{L}}$ are matrices $\mathbf{K}$ and $\mathbf{L}$ respectively, with diagonal entries set to 0, $\bar{\mathbf{A}}_{ij} = (1 - \delta_{ij})\mathbf{A}_{ij}$.

The choice of the linear kernel is motivated by the study of Kornblith et al. (2019), who studied the similarity between neural network representations and showed little difference to non-linear kernels, such as the Radial Basis Function kernel (RBF).

While Centered Kernel Alignment allows studying the similarity between two distinct heads, it does not measure the redundancy of each head directly. To develop a metric of head redundancy, I compute a pairwise similarity matrix $\mathbf{S}$ for the set of all heads i, j within each layer. The entries of $\mathbf{S}$ are defined as:

$$\mathbf{S_{ij}} = \text{CKA}\left(softmax\left(\frac{\mathbf{X}\mathbf{W}_\mathbf{Q}^{(\mathbf{i})}(\mathbf{X}\mathbf{W}_\mathbf{K}^{(\mathbf{i})})^T}{\sqrt{d_k}}\right), softmax\left(\frac{\mathbf{X}\mathbf{W}_\mathbf{Q}^{(\mathbf{j})}(\mathbf{X}\mathbf{W}_\mathbf{K}^{(\mathbf{j})})^T}{\sqrt{d_k}}\right)\right). \quad (4.15)$$

Note that:

1. the diagonal of $\mathbf{S}$ is filled with 1s, indicating maximum self-similarity of each head,
2. and since CKA is symmetric, $\mathbf{S}$ is also symmetric.

Although computing the pairwise similarity matrix scales quadratically with the number of heads, the above observations help decrease the total computation by more than half.

Having computed the pairwise similarity matrix for each layer of the model, I introduce four additional scalar metrics based on $\mathbf{S}$, which are logged throughout the training process over time, and help understand the training dynamics. The first two summarise attention within each layer:

- **Mean redundancy per layer**, R_{mean}, defined as a simple average of the upper triangular **S**, excluding the diagonal. This metric summarises the average amount of similarity between attention heads in each layer and is
- **Max redundancy per layer**, R_{max}, defined as the maximum entry of the upper triangular **S**, excluding the diagonal. It helps discover if there exists a head pair with a particularly high similarity, possibly suggesting redundancy.

The second group is designed to track attention on the more granular, head-level:

- **Mean head redundancy for head** i, $R^{(i)}_{mean}$ defined as a simple average of the ith row of S, excluding the diagonal entry. Higher values indicate more similarity with other heads, and possibly lower utility of a given head.
- **Max head redundancy**, $R^{(i)}_{max}$, defined as a maximum entry of the ith row of S, excluding the diagonal entry. This final metric directly displays the highest similarity to any other head.

I further aggregate mean head redundancies across all heads and layers, and represent them as **head redundancy maps**. This method allows for the study of the change of attention head redundancy for the entire model throughout the training. Values close to 1 indicate relatively high redundancy of a given head. Smaller values suggest that different heads capture more distinct patterns. Interestingly, some highly-redundant layers include outliers – heads capturing particularly unique patterns, different from all other heads.

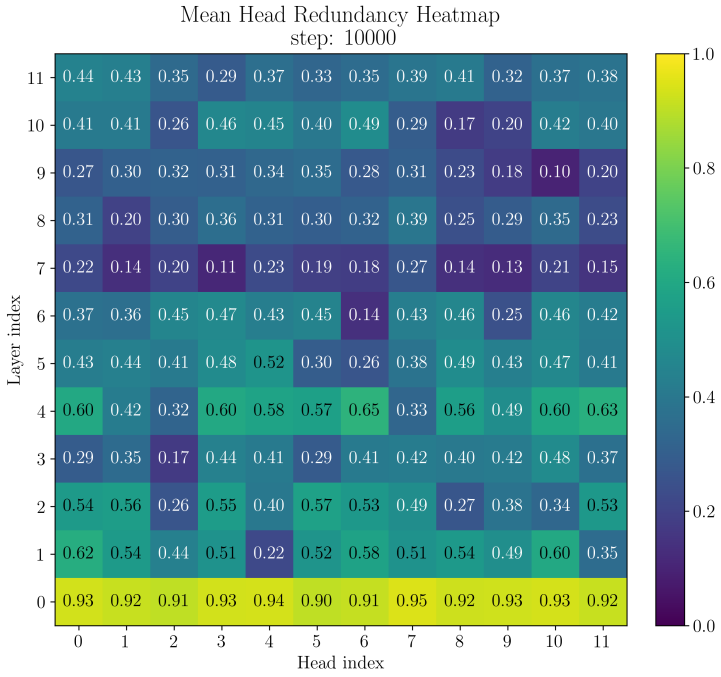


Figure 4.5: Head redundancy map of a trained 124M GPT model. Note that head redundancy decreases with the depth of the model up until layer 7. Interestingly, all heads in the initial layer remain relatively similar after training on Chinchilla compute-optimal number of tokens.

FLOPs Count and Training Time

In addition to tracking the language modelling ability or specifics of the attention mechanism, I report the training wall-time and the estimated number of floating point operations (FLOPs). To estimate FLOPs, I use the methodology proposed by Chowdhery et al. (2022), who assume the number of floating point operations per token can be split into those coming from dense feedforward layers, and those coming from the self-attention computation. The former can be approximated to require $6N$ FLOPs per token, where N is the total number of model parameters. Computing the number of FLOPs in attention depends on the number of heads m, their dimension d_k, and the context length T, and is given by $12md_kT$ for each of the L layers. Given that SMHA allows a variable number of heads in each layer, the number of FLOPs per token is estimated as:

$$FLOPs_{actual} = 6N + \sum_{l=1}^{L} 12m^{(l)}d_kT,$$

where $m^{(l)}$ is the number of attention heads in the lth layer of the model.

4.3.3 Static vs Dynamic Attention Ratio

In addition to the pre-defined attention ratio in each layer, it is possible to dynamically modify it during training. This can aid with training instability, particularly visible in the initial stage of large model optimisation. Starting with a less expressive model and gradually increasing its capacity may lead to more stable optimisation and better convergence. Early experimental results consistently displayed a dependence between redundancy metrics and the attention ratio.

To determine when to increase the attention ratio for a particular layer, I use the max redundancy per layer R_{max} as a proxy to quantify the need for more expressivity in the model. Measuring the highest level of redundancy within a layer, this metric can be compared against a pre-defined **redundancy threshold** θ_R. Intuitively, the redundancy of a layer over the threshold means that attention heads are not yet diverse enough, hence the attention ratio should remain unchanged, or possibly be lowered. If the redundancy of a layer is lower than the threshold, the attention heads capture different patterns, suggesting a need for more expressivity.

$$R_{max} \begin{cases} \geq \theta_R, \text{layer is expressive enough, keep } \alpha^{(l)} \text{ unchanged} \\ < \theta_R, \text{layer needs more expressivity, double } \alpha^{(l)} \end{cases}$$

Since R_{max} is expected to change over time, the threshold can be dynamically adjusted as well. In the early stage of training, the threshold should be lower to reflect the need for more diversity and to limit the number of trainable parameters. With the training

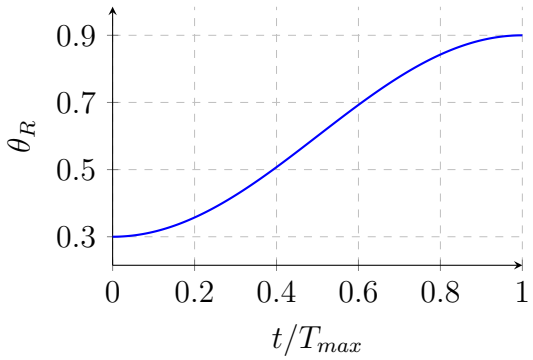


Figure 4.6: Schedule of the redundancy threshold θ_R throughout training. Initial and final values of $\theta_0 = 0.3$ and $\theta_T = 0.9$, respectively.

progressing, the threshold can be gradually increased to allow for more attention heads and redundancy between them. Furthermore, to achieve initial model convergence, the threshold should change more slowly at the start, and vary less towards the end, when added heads may not receive enough updates to initialise. Hence, to vary the redundancy threshold throughout training, I use an inverted cosine annealing scheduler, defined as:

$$\theta_R = \theta_0 + (\theta_T - \theta_0) \times \frac{1}{2}\left(1 - \cos\left(\frac{\pi t}{T_{max}}\right)\right),$$

where t is the current training iteration, T_{max} is the total number of iterations, θ_0, and θ_T are the initial and final threshold values, respectively. To prevent the system from doubling the attention ratio for the same layer in two consecutive iterations, I add a per-layer cooldown period of 10% of training iterations after modifying the layer. Additionally, to account for the initialisation of the model, no attention heads are added in the first 10% of training. An example of the cosine threshold schedule is presented in Figure 4.6.

Dynamically increasing the number of attention heads during training results in additional challenges. Since the added attention heads have not been trained before, their random initialisation may introduce instability to the training and lead to divergence of the model. To remedy this, Xia et al. (2023) suggested using the weights of the already trained heads to initialise the new ones. On the other hand, many of the current LLM training frameworks leverage learning rate schedulers with annealing, such as the cosine annealing scheduler. While decreasing the learning rate stabilises training, it may lead to additional undertraining of attention heads added later in the training process. Hence, I propose to use a warmup-stable-decay (WSD) scheduler, which allows to maintain a constant learning rate value for the majority of the training and decay it in the final iterations. Note, however, that this method only partially addresses the issue of undertraining, since parameters added in the middle of the optimisation trivially receive fewer updates. Both the attention head initialisation and the learning rate scheduling are explored in the ablation studies further in this dissertation.

4.3.4 SMHA in Heterogeneous FL

Applying Selective Multi-Head Attention to the hardware-heterogeneous FL has multiple use-cases. It allows for a decrease in client round times by limiting the amount of computation per client. Furthermore, it allows for better control of the idling in the system. In the case when strategy 3 is used, it may help to decrease the difference in the number of local optimisation steps between clients. Finally, depending on the implementation, it may lead to a slight decrease in the memory usage and the amount of communicated data when some of the attention heads are not trained on a given client.

SMHA allows training only some parts of the model, leading to a decreased training time. Therefore, I consider a case, when each of the C clients trains only a fraction of the available attention heads n_{head} but they **jointly aim to optimise the complete model**. Note, however, that the evaluation of the local and aggregate models differs fundamentally – the perplexity of the local models PPL_{post} is computed using only the trained attention heads, while the aggregated model PPL_{pre} is evaluated using all of them.

Decreasing Client Round Times

To simplify this setup, I assume that the indices of trainable heads do not change across the model, and the selected attention head set I_h is identical between layers. Formalising it further, for each head index i, I define a **head training ratio** $\lambda^{(i)}$, as the ratio between the number of assigned clients to the total number of clients. Two distinct scenarios arise:

- Head i is trained by only one client per round ($\lambda^{(i)} = \frac{1}{n_{head}}$).
- Head i is assigned to n clients per round ($\lambda^{(i)} = \frac{n}{n_{head}}$).

Similarly to α_{attn}, when upper index is omitted, the value of λ applies to the entire model.

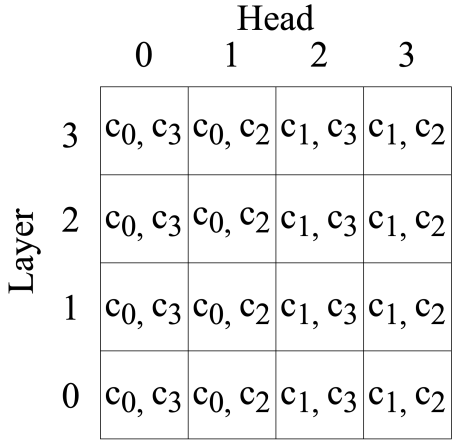


Figure 4.7: SMHA-equipped 16M-4 model trained jointly by many clients, with $\lambda = 0.5$ for all heads. Note that in this example, no single client trains the entire model, yet all attention heads are trained twice.

From the optimisation perspective, the first case is more difficult because the head parameters are not averaged across clients. However, it also minimises the total amount of

computation performed by the federation. Hence, I first study how different values of λ affect the convergence and training time in the hardware-homogeneous setting. Subsequently, I apply it to the hardware-heterogeneous case. In these experiments, I assume a constant α_{attn}.

Distributing Heads Across Clients

Assigning heads to clients can be performed either only once at the start of the training or dynamically changed every round. When clients train more than a single head, it may be important to dynamically change which heads are trained together. While head assignment neither changes their λ nor affects the client compute time, it may lead to differences in the final model performance. Therefore, in the last experiment, I explore which of the two assignment methods results in lower perplexity. Similarly, in these experiments I assume a constant α_{attn}.

Chapter 5

Experimental Setup & Design

5.1 Experimental Setting

In all three experimental sections in this work, I pre-train language models based on the GPT architecture. Due to the limited resources and significant energy usage of pre-training, most of the experiments and ablation studies utilise a smaller model with 16 million parameters (16M). This model comes in two flavours: with 4 attention heads (16M-4) and 8 attention heads (16M-8). Note that the hidden size of the model is identical between the two, changing the dimensions of the attention key and value d_k, d_v only. The final experiments demonstrating the scaling of the studied methods are performed using a larger model with around 124 million parameters (124M). The hyperparameters of both models are presented in Table 5.1.

Hyperparameter	**16M-4**	**16M-8**	**124M**
Number of layers (n_{layer})	4		12
Hidden size (d_{model})	256		768
Expansion ratio	4		
Number of heads (n_{head})	4*	8*	12*
Attention head key size (d_k)	64	32	64
Attention head value size (d_v)	64	32	64
Vocabulary size ($\|\mathcal{V}\|$)	50368		
Max sequence length (seq_{max})	512		2048
Compute-optimal token count	$\sim$ 320M	$\sim$ 320M	$\sim$ 2480M

Table 5.1: Model hyperparameters of both models. *Note that the number of attention heads represents the maximum available number of heads. The actual number of heads used in an iteration may be lower when SMHA with $\alpha_{attn} < 1$ is used.

I use different data for model training and evaluation between, depending on the experiments section. For experiments studying the Selective Multi-Head Attention in the centralised setting, I use the open-source replication of the OpenWebText dataset (Gokaslan and Cohen, 2019), split into a training and validation set. For the other experiments, I

use the IID-partitioned train and validation splits of the English subset of the C4 dataset (Raffel et al., 2023).

5.1.1 Training Budget, Server Rounds and Local Steps

Pre-training foundational models falls into the large data regime, where the model performance improves with the amount of data the model is trained on. Hence, it is essential to establish the training budget. Conveniently, in the context of LLM training, Hoffmann et al. (2022) experimentally showed the dependence between the optimal number of training tokens and the model size. Hence, I use the scaling laws from the *Approach 1* of their work and estimate the model size and token count following the inferred power laws proportional coefficients to be $\alpha = 0.0913$ and $\beta = 1.8257$. This results in the following relationship between the optimal number of training tokens D_{opt} and the optimal number of model parameters N_{opt}:

$$D_{opt} = \frac{\beta}{\alpha} \cdot N_{opt} \tag{5.1}$$

$$D_{opt} = 19.9967 \cdot N_{opt} \tag{5.2}$$

$$D_{opt} \simeq 20 \cdot N_{opt}. \tag{5.3}$$

While the number of server rounds R is an important FL hyperparameter, I do not focus on its optimisation in this work. The number of rounds is dependent on the ratio between the length of a computation round and the subsequent communication period. The latter is lower-bounded by the available bandwidth between clients. When FL is used in the cross-silo setting to combine the computational resources and allow large model training, it may be desired to select R such that it outperforms the wall-time of centralised training. This is achieved when throughout the training the clients communicate $\frac{B_C}{B_{FL}}\times$ less frequently, where B_C and B_{FL} represent respectively the bandwidth between data-parallel accelerators used in a centralised setting, and the bandwidth between the clients.

5.1.2 Implementation

All experimental and analysis code has been written in Python using PyTorch. The experimental setup of the SMHA centralised study is based on the heavily modified *nanoGPT* repository[1], which provides very granular access to components involved in model pre-training. This allows studying how the choice of model hyperparameters affects the model training dynamics.

The federated learning experiments are based on the adapted *Photon* codebase, an advanced state-of-the-art federated learning framework that allows multi-node, multi-GPU

[1] `https://github.com/karpathy/nanoGPT`

training. The adaptations involved adding the possibility of simulating hardware-heterogeneous clients with varying hyperparameters and tracking additional metrics. Some external libraries, on which Photon relies, also required modifications. Notably, the GPT implementation in the *llmfoundry* library[2] has been adjusted and updated to include the SMHA implementation. Furthermore, the gradient noise scale measuring callback has been modified in the *composer* library[3]. The exact listing of code repositories and changes made is available in Appendix A.

5.2 Experiment Design

5.2.1 Hardware-heterogeneous FL

All centralised and FL experiments are simulated on a local cluster. The FL experiments are performed using the Photon system with a modification which allows for granular control over per-client hyperparameters. In all FL experiments, I choose the FedAvg aggregation strategy with learning rate 1, and simulate different maximal compute capabilities of clients (further referred to as *client types*[4]) by constraining the available hardware (8× NVIDIA A40 48GB GPUs). For each client type, I define the value of B_{max} and experimentally measure the t_{bmax} for the studied model sizes (using complete Multi-Head Attention). Using the obtained values, I define two different configurations of hardware-heterogeneous client types, defined in the Table 5.2:

- Configuration 1 consisting of 2 clients: $\{c_0 = C_1,\ c_1 = C_2\}$
- Configuration 2 consisting of 4 clients: $\{c_0 = C_1,\ c_1 = C_2,\ c_2 = C_2,\ c_3 = C_3\}$

Furthermore, I consider two additional hardware homogeneous configurations:

- Configuration 3 consisting of two clients C_1
- Configuration 4 consisting of four clients C_1

In all federated experiments, I assume a constant number of federated rounds $R = 38$.

	16M	
Client Type	$B_{\max}$	$t_{b_{\max}}$
C_1	32	0.165s
C_2	16	0.129s
C_3	8	0.112s

Table 5.2: Comparison of $B_{\max}$ and $t_{b_{\max}}$ across client types

[2] `https://github.com/mosaicml/llm-foundry`
[3] `https://github.com/mosaicml/composer`
[4] Note, that client types are marked C_i, different from client instances c_i.

Learning Rate Search

To explore the influence of the learning rate value on training, I perform a grid search over the following six values: {1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 5e-3}. To better understand how to translate the optimal learning rate values between the centralised and federated settings, and how they relate to mini-batch sizes, I study the convergence of: a hardware-homogeneous *Configuration 4* run with a per-client mini-batch size of $\frac{B_{eff}}{C}$, a centralised run with a mini-batch size of B_{eff}, and a centralised run with a mini-batch size of $\frac{B_{eff}}{C}$.

In all runs, I use the 16M-4 models and perform $L = 4880$ optimisation steps. Given that the optimal B_{eff} and the optimal number of local steps $\frac{L}{R}$ is unknown at this point, I set them arbitrarily to 128 and 128, respectively. This results in the total of $S_{16M} = 624,640$ processed samples.

GNS in the FL Setting

To investigate whether Gradient Noise Scale can be used in the federated setting, I perform two training runs for each model size: one following a centralised setting and one using hardware-homogeneous *Configuration 4*. In both cases, I ensure that the total number of iterations steps, processed samples and hence the effective mini-batch size B_{eff} is equal, and set it similarly to the previous experiment ($L_{16M} = 4880$, $S_{16M} = 624,640$, $B_{eff} = 128$ and $L_{124M} = 4880$, $S_{124M} = 1249280$, $B_{eff} = 256$). Following the $\mathcal{B}_{noise}$ estimation methodology using data parallelism, I set ($B_{big} = B_{eff}$, $B_{small} = 16$) for centralised experiments. Since in the FL setting GNS is estimated on the per-client basis, I configure clients with ($B_{big} = \frac{B_{eff}}{C}$, $B_{small} = 16$). The learning rates are set to the optimal values of the previous study.

Heterogeneity Resolution Strategies

In this section, I compare the training wall-time, total idling time in the system and model perplexity between different strategies. I assume a fixed training budget T, the number of server rounds R, and setting the number of samples/round constant. Combining the results from the previous experiments with the empirically measured client capabilities, I study all hardware-heterogeneity resolution strategies, with hyperparameters computed following the methods described in Section 4.2. To calculate the number of samples per round, I use $\frac{T}{seq_{max} \times R}$, where R is set to 38. While all experiments are performed using the 16M-4 model, I set T to half the compute-optimal number of tokens for the two-client *Configuration 1* and the full compute-optimal number of tokens for the four-client *Configuration 2*. With some rounding for divisibility, for the first configuration I choose 8,192 samples per round, and for the second, 16,384, due to their divisibility. The exact hyperparameters for all strategies are presented in Table 5.3 for *Configuration 1* and Table 5.4 for *Configuration 2*.

Note that, depending on the available mini-batch sizes, it may be impossible to arrive at

the target number of samples per round (e.g. Strategy 1). In those cases, I aim to get to the closest possible number of samples per round.

c_i	Client type	Local steps L	Mini-batch size	Samples/round	LR
Strategy 1 – total samples/round: 8,208					
0	C_1	171	32	5,472	5.0e-4
1	C_2	171	16	2,736	2.5e-4
Strategy 2a – total samples/round: 8,192					
0	C_1	128	32	4,096	5.0e-4
1	C_2	128	32	4,096	5.0e-4
Strategy 2b – total samples/round: 8,192					
0	C_1	128	32	4,096	5.0e-4
1	C_2	256	16	4,096	2.5e-4
Strategy 3 – total samples/round: 8,192					
0	C_1	156	32	4,992	5.0e-4
1	C_2	200	16	3,200	2.5e-4

Table 5.3: Comparison of the resolution strategies hyperparameters for the 16M-4 model in Configuration 1.

c_i	Client type	Local steps L	Mini-batch size	Samples/round	lr
Strategy 1 – total samples/round: 16,344					
0	C_1	228	32	5,472	5.0e-4
1	C_2	228	16	3,648	2.5e-4
2	C_2	228	16	3,648	2.5e-4
3	C_3	228	8	1,824	1.2e-4
Strategy 2a – total samples/round: 16,384					
0	C_1	128	32	4,096	5.0e-4
1	C_2	128	32	4,096	5.0e-4
2	C_2	128	32	4,096	5.0e-4
3	C_3	128	32	4,096	5.0e-4
Strategy 2b – total samples/round: 16,384					
0	C_1	128	32	4,096	5.0e-4
1	C_2	256	16	4,096	2.5e-4
2	C_2	256	16	4,096	2.5e-4
3	C_3	512	8	4,096	1.2e-4
Strategy 3 – total samples/round: 16,384					
0	C_1	193	32	6,176	5.0e-4
1	C_2	248	16	3,968	2.5e-4
2	C_2	248	16	3,968	2.5e-4
3	C_3	284	8	2,272	1.2e-4

Table 5.4: Comparison of the resolution strategies hyperparameters for the 16M-4 model in Configuration 2.

5.2.2 SMHA Analysis in the Centralised Setting

Hardware-heterogeneous federated learning has significantly more hyperparameters and contributes more confounding factors to experiments. Hence, before employing SMHA in that setting, I begin by extensively studying SMHA in a centralised scenario. The analysis is performed to understand how varying the number of attention heads affects the training dynamics.

The training hyperparameters for both the 16M-8 and 124M models are shown in Table 5.5. No dropout is used, and the warmup-stable-decay (WSD) scheduler is employed for training extensibility.

Training Hyperparameter	**16M**	**124M**
Learning rate (η)	1e-3	6e-4
Micro-batch size (B)	32	16
Gradient accumulation steps	5	20
Mini-batch size (B)	160	320
Number of training steps	4,000	10,000
Warmup steps	400	500
Decay steps	800	2,000
Optimizer	AdamW	
Learning rate schedule	Warmup-Stable-Decay (WSD)	
Fraction of compute optimal budget	1.02	1.32

Table 5.5: Training hyperparameters for 16M and 124M GPT models in SMHA experiments.

Experiments

I start by studying how head redundancy metrics and similarity matrices change throughout training in a unmodified Multi-Head Attention model. Following this study, I train a set of models with a set of lower static attention head ratios α_{attn} (Table 5.6), comparing their perplexity and attention-related metrics with the original models. In each training run, the number of attention heads is kept equal in every layer and not modified during training.

α_{attn}	**16M**	**124M**
$\alpha_{attn} = 0.25$	2/8	4/12
$\alpha_{attn} = 0.5$	4/8	6/12
$\alpha_{attn} = 0.75$	6/8	8/12
$\alpha_{attn} = 1$	8/8	12/12

Table 5.6: Attention head ratios and the corresponding head counts.

Furthermore, I investigate the effect of dynamically modifying the attention head ratio during training. To do so, I initialise a model with the $\alpha_{attn} = 0.25$ and grow each layer

individually, only when its R_{max} is lower than the redundancy threshold θ_R, and outside of the cooldown period. The threshold is scheduled using the inverse cosine scheduler, with $\theta_0 = 0.4,\ \theta_T = 0.9$ for 16M-8, $\theta_0 = 0.3,\ \theta_T = 0.9$ for 124M.

This study involves two additional ablation experiments. In the first one, I investigate the influence of the WSD scheduler on the dynamic changes of α_{attn}. I aim to verify the hypothesis, whether when new attention heads are added to the model, the learning rate needs to be kept relatively high to prevent them from under-training. Hence, I compare the WSD scheduler to a Cosine Annealing scheduler, applied to the dynamically changed α_{attn}. The last study aims to verify whether the newly added heads can be initialised from the random state (Gaussian distribution initialisation), or should be initialised by copying the parameters of the already trained heads, as suggested by (Xia et al., 2023).

5.2.3 SMHA in Heterogeneous FL

The experiments performed in this section make use of the same configurations and client types as defined in Section 5.2.1. In the first experiment, I compare the following two *Configuration 4* training runs:

- An unmodified 16M-4 model with $\alpha_{attn} = 1$ and $\lambda = 1$.
- A 16M-4 model with $\alpha_{attn} = 0.25$ and $\lambda = 0.25$, with head i assigned to a distinct client i, $(c_i : \{i\})$.

In the follow-up experiment, I further compare two Configuration 4 runs of a 16M-4 model, where every client's α_{attn} is set to 0.5 and $\lambda = 0.5$. Originally, I assign head ids to clients in the following way: $(c_0 : \{0, 1\},\ c_1 : \{2, 3\},\ c_2 : \{0, 1\},\ c_3 : \{2, 3\})$. Then, two cases are compared to check whether it is necessary to change which heads are trained together throughout the training:

- In the first case, head assignments are static during training.
- In the second case, I modify the head assignments every round, changing between all possible head pairings, ensuring every head is trained the same number of times:
  - $(c_0 : \{0, 1\},\ c_1 : \{2, 3\},\ c_2 : \{0, 2\},\ c_3 : \{1, 3\})$,
  - $(c_0 : \{0, 3\},\ c_1 : \{1, 2\},\ c_2 : \{0, 1\},\ c_3 : \{2, 3\})$,
  - $(c_0 : \{0, 2\},\ c_1 : \{1, 3\},\ c_2 : \{0, 3\},\ c_3 : \{1, 2\})$.

Finally, I apply SMHA to Configuration 2 and strategy 3., where client types are associated with the following α_{attn} values: $C_1 : \alpha_{attn} = 1$, $C_2 : \alpha_{attn} = 0.5$, $C_3 : \alpha_{attn} = 0.25$. Furthermore, I implement a dynamic head assignment scheme. Note that, in this arrangement, the system in total can train 9 heads per round, where each head is trained exactly twice on the first three clients c_0, c_1, c_2 ($\lambda = 0.5$). However, the fourth client trains one of the heads additionally, leading to ($\lambda \geq 0.5$).

Chapter 6

Results

6.1 Hardware-Heterogeneous FL

6.1.1 Learning Rate

Centralised Learning Rate Cannot be Easily Translated Into FL Setting. To understand how to choose learning rate for federated setting with effective mini-batch size B_{eff} and per-client mini-batch size $\frac{B_{eff}}{C}$, I studied how model perplexity compares to centralised settings with effective batch size equal to either B_{eff} or $\frac{B_{eff}}{C}$. As shown in Figure 6.1, the federated run does not follow either of the centralised settings. Interestingly, it is possible to distinguish two regimes: with lower learning rates (1e-5, 5e-5), the federation achieves comparable results to the centralised case with the equivalent effective mini-batch size. However, when the learning rate grows (1e-4), the final validation perplexity gets closer to the centralised case with the batch size equivalent to the batch size of federate clients. Finally, with significantly larger learning rates (5e-4, 1e-3, 5e-3), the centralised approaches outperform the federation.

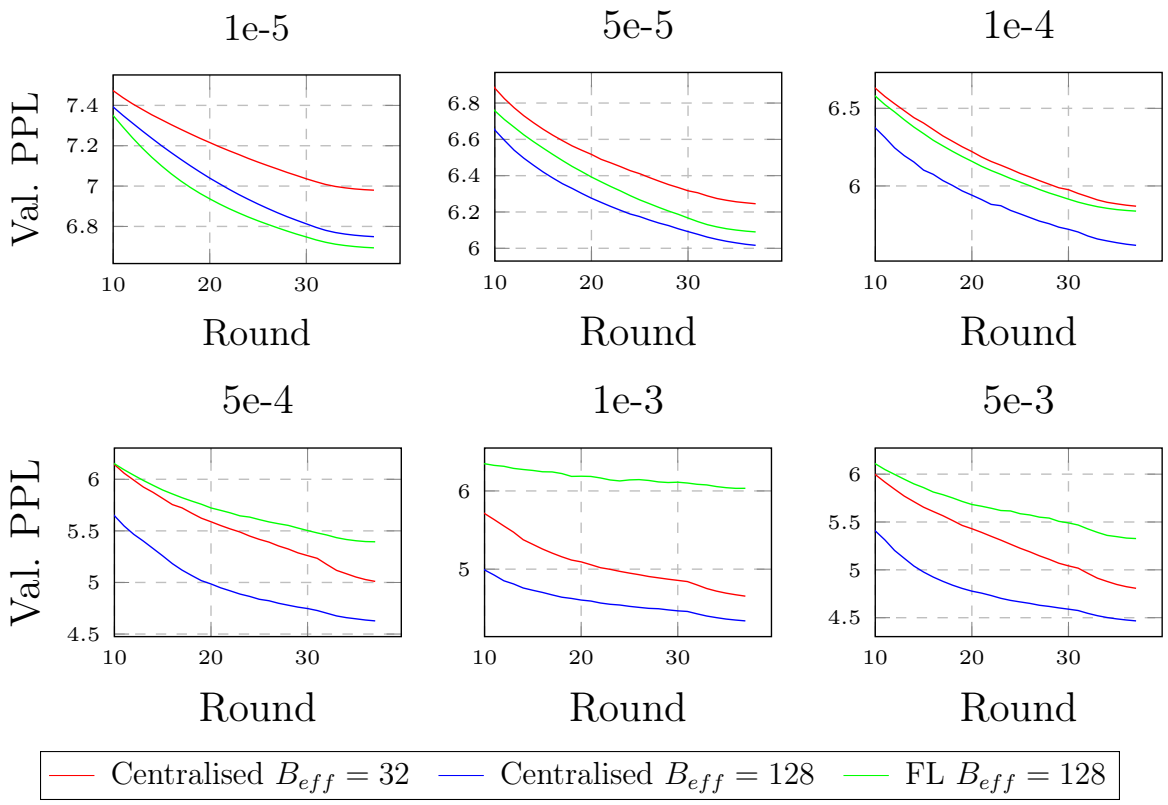


Figure 6.1: Learning rate sweep and validation perplexities between centralised and federated runs. Note that the convergence rate of the federated setting is progressively diverging from centralised baselines with growing learning rate.

This behaviour could suggest that federated averaging leads to degradation of pseudo gradient contributions when individual clients explore the loss landscape more aggressively. One way to explore this idea further is to study the dynamics of the gradient noise scale $\mathcal{B}_{simple}$, which may indicate whether the issue lies in the size of a mini-batch. Since it is not possible to reliably predict learning rate for hardware-heterogeneous clients based on the centralised setting, in further studies I use the empirically obtained optimum in the FL case (1e-4), and linearly scale it with mini-batch size.

6.1.2 Gradient Noise Scale in Federated Learning

Gradient Noise Scale of Federated Clients Does Not Follow Centralised Dynamics. Understanding the relation between GNS in centralised and federated settings is useful, because if they are similar, then the already developed theory for the former setting could be applied to tune mini-batch size in the hardware-heterogeneous FL. However, as shown in Figure 6.2, $\mathcal{B}_{simple}$ behaves differently in both cases.

I start this analysis by focusing on the central phase of the learning rate schedule (marked with dashed lines). Regular aggregation and averaging of weights in federated learning seem to be inhibiting the growth of $\mathcal{B}_{simple}$ experienced in the centralised study. As explained by McCandlish et al. (2018), the growth is expected and comes from the progressively decreasing gradient direction, minimising the loss, and more background noise.

No similar pattern in any of the FL runs has been observed, which may suggest that parameter aggregation acts as a “reset" point, providing clients with an optimisation task, which has a more distinct gradient direction, minimising the loss. To explore this idea further, I zoom in on the gradient noise scale curves in Figure 6.3, which shows a slight oscillatory pattern in the value of $\mathcal{B}_{simple}$. The period of the pattern coincides with parameter aggregation points, supporting the reasoning about its source. Furthermore, after each synchronisation point (every 128 steps), $\mathcal{B}_{simple}$ drops and starts increasing again. Note that while it does not drop immediately in plots shown, it is likely because of the applied EMA averaging, as explained in the methodology of estimating $\mathcal{B}_{noise}$.

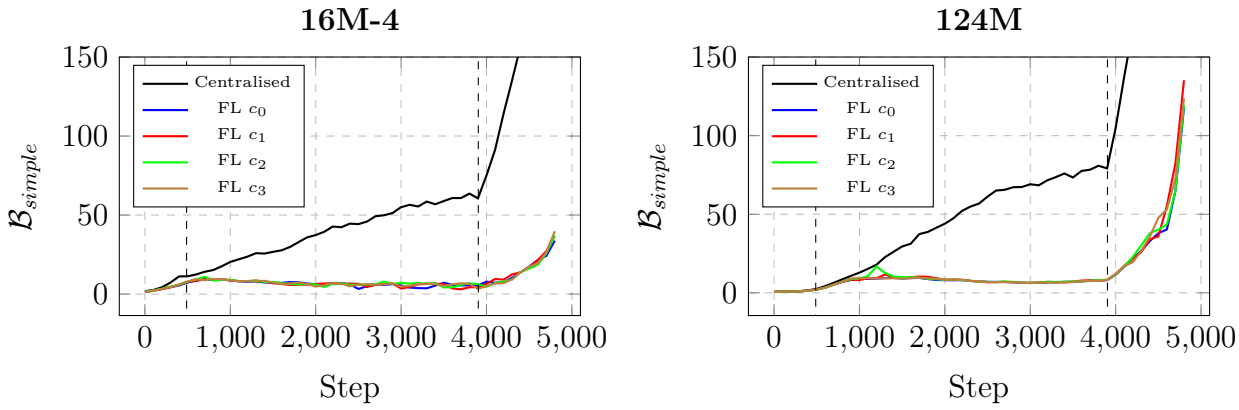


Figure 6.2: Gradient Noise Scale estimation $\mathcal{B}_{simple}$ comparison between centralised and hardware-homogeneous FL settings. Models trained with Chinchilla compute-optimal training budgets.

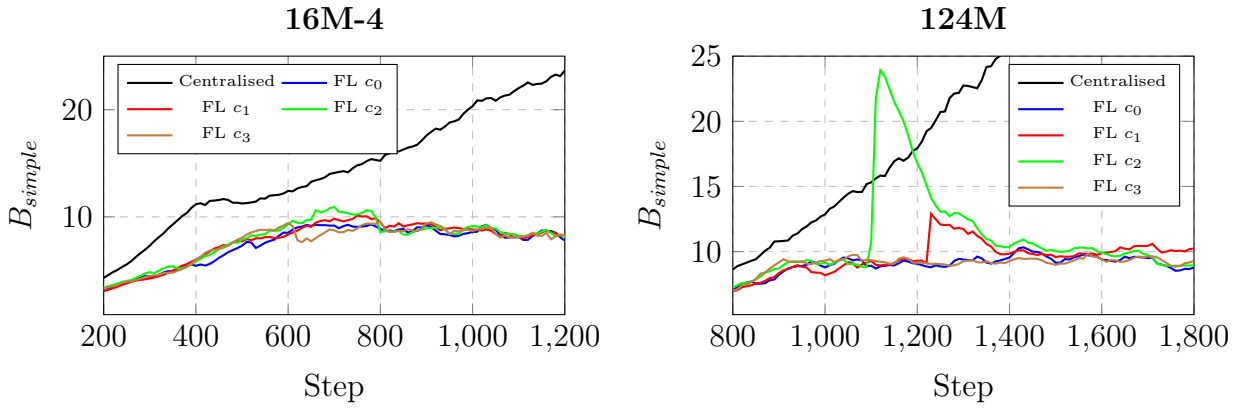


Figure 6.3: Zoomed-in $\mathcal{B}_{simple}$ curves at the divergence region.

On the other hand, there is a clear divergence point between $\mathcal{B}_{simple}$ of the centralised and federated run. For both studied model sizes, it occurs early, after a similar number of rounds. Figure 6.3 shows those divergence regions, indicating that in the case of the smaller model, it occurs around step 800 (after the 6th round), and for the larger model, the divergence takes place around step 900 (after the 7th round). While the location of the divergence could be dictated by the changing value of the learning rate, the change of the scheduler phase from warm-up to stable occurs significantly earlier, at step 488.

Federated Learning May Have Smaller Compute-Optimal Mini-Batch Size. Measuring the $\mathcal{B}_{simple}$ is a method used in estimating the compute-optimal mini-batch size B_{crit}. Hence, if the reasoning about the per-round aggregation as a "reset" for local optimisation is correct, then it suggests that federated learning with FedAvg aggregation strategy may have a lower value of B_{crit}, resulting in either being a more computationally efficient method or less scalable. However, verifying this hypothesis requires further experiments, especially studying how the gradient noise scale reacts to varying the number of local optimisation steps and changing the federated aggregation strategy.

Warmup-Stable-Decay Scheduler has three $\mathcal{B}_{simple}$ Regimes. The observed values of $\mathcal{B}_{simple}$ fall into three regimes, which are marked with the vertical dashed lines and indicate the phases of the learning rate scheduler. Note that in the original study by McCandlish et al. (2018), the authors do not consider the effects of dynamically varying learning rate during training. This adds complexity to estimating B_{crit} because the value of $\mathcal{B}_{simple}$ itself depends on the chosen learning rate. Hence, it is reasonable to split estimation of the B_{crit} into the three learning rate scheduling phases, as shown in Table 6.1. Since the stable phase is the longest, I use the mean value of $\mathcal{B}_{simple}$ from this period as the compute-optimal mini-batch size in further experiments. Furthermore, I use the results from the centralised runs with the expectation, that the B_{eff} in the centralised case is equivalent to the B of each client in the federated scenario. Finally, I set $B_{crit} = 32$ per client for 16M-4 and $B_{crit} = 64$ per client for 124M.

Case	**Total**	**Warmup**	**Stable**	**Decay**
		16M-4		
Centralised	74.3	6.2	39.4	230.3
FL client avg	8.30	3.6	6.7	16.2
		125M		
Centralised	106.8	1.1	47.0	368.9
FL client avg	14.4	1.0	7.8	44.9

Table 6.1: Measured mean values of $\mathcal{B}_{simple}$ for different scheduler phases.

6.1.3 Heterogeneity Resolution Strategy

Heterogeneous-Hardware Resolution Strategy 3 Decreases the Total Idling Time Significantly. Looking at the results of the four-client configuration 2, different strategies show vastly different training wall-times and idling ratios. Due to the inevitable idling of either the slower clients (strategy 1) or the faster clients (strategies 2a and 2b), the training wall-time of the federation is adequately extended. The differences in round processing times for the participating clients can be seen in Figure 6.4. These differences result in an average client inactivity ratio between 20% and 38%. In a practical FL deployment, client owners may decide to run external workloads during the

idle time, yet it adds complexity to maintaining client state and results in significantly longer federation training time. The same pattern repeats in the case of the two-client configuration 2, with smaller idling ratios due to the smaller number of clients.

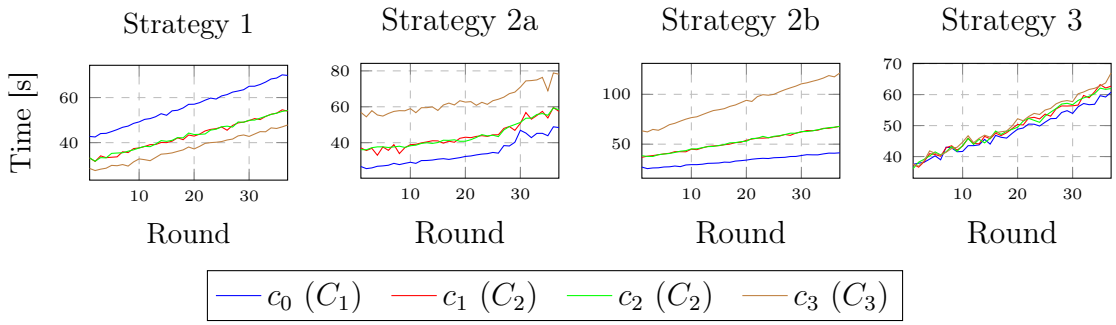


Figure 6.4: Round processing times for each of the four clients, across different strategies. Note that the growing trend results from a bug in the implementation of the original library, yet scales the per-client times identically, hence does not affect the comparison.

Empirically measuring throughput of each client and adjusting the number of optimisation steps, as suggested by strategy 3, leads to a significant reduction of the idle time, not far from the hardware-homogeneous case (2.62% vs 1.64% in configuration 2). In the strategy 3 experiment, I use a constant t_{bmax} across all rounds, however, when client throughput is expected to change, t_{bmax} and client optimisation step counts may be dynamically adjusted.

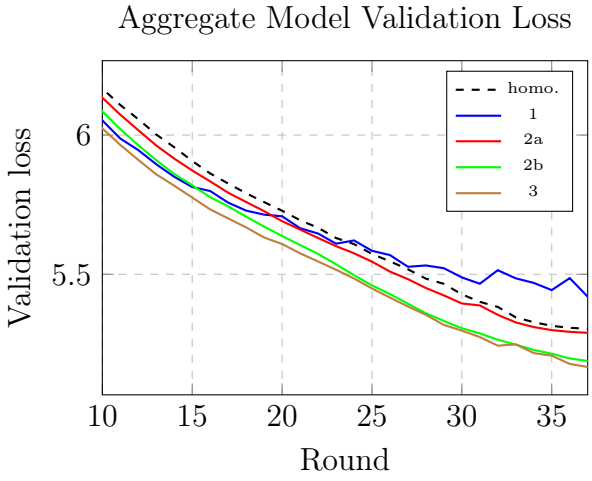


Figure 6.5: Validation loss for each of the strategies considered. Both strategies 2b and 3 achieve better results than the hardware-homogeneous run with high-performance clients. This indicates that an increased number of steps leads to better convergence.

Strategy 3 Achieves Lower Perplexity Than Homogeneous Case with High-Performance Clients. Limiting the idle time through local step adjustments is successful only if it does not lead to a significant decrease in model performance. Table 6.2 shows that most of the heterogeneous runs result in lower final validation perplexity than the homogeneous-hardware baselines. Similarly, Figure 6.5 shows that these strategies are consistently better throughout the duration of the training. This means that with the

fixed training budget and the number of rounds, less capable clients contribute to the aggregated model more than high-performing ones, by taking more optimisation steps, even with a smaller batch size and adequately adjusted learning rate. Crucially, in the case of the four-client configuration 2, strategy 3 achieves the lowest perplexity, significantly outperforming the homogeneous-hardware run.

Strategy	**Training Wall-Time**	**Avg. Idling Ratio/client**	**PPL**
1	2,078s	20.11%	5.454
2a	2,344s	26.85%	5.290
2b	3,376s	37.10%	5.238
3	**1,876s**	**2.62%**	**5.185**
homo.	1,318s	1.64%	5.315

Table 6.2: Training time, average idling ratio per client and final model perplexity for the four-client hardware-heterogeneous configuration.

Notably, configuration 2 is equipped with hardware allowing almost only half (56%) of the effective mini-batch size of the four-client homogeneous configuration. Nevertheless, using strategy 3 required only 42% more time to complete. This promising result may indicate that there is an optimisation benefit to aggregating parameters of models trained with different numbers of steps. On the other hand, it may suggest that learning rates across configurations were not configured in the optimal way, giving some of the clients an unfair advantage. Regardless, both possibilities indicate plausible gains of heterogeneous-hardware FL.

In Strategy 1, Top-Performing Client Dominates the Federation. The amount of contribution each client brings to the aggregated model changes throughout the duration of training and depends on the chosen heterogeneity-resolution strategy. The measured client gap values for each client c are presented in Figure 6.6. The client gap ratio below 1 of the top-performing client c_0 in strategy 1, indicates that its contributions are better than the aggregated model. This indicates that when clients perform the same number of optimisation steps, yet with different mini-batch sizes and linearly scaled learning rate values, the most capable of them dominates the federation. This may also imply that the federation slows down the optimal convergence, which would be achieved if client c_0 performed optimisation on its own. Furthermore, this reasoning has support in the significantly lower perplexity of strategy 1 and the corresponding shape of the loss curve in Figure 6.5.

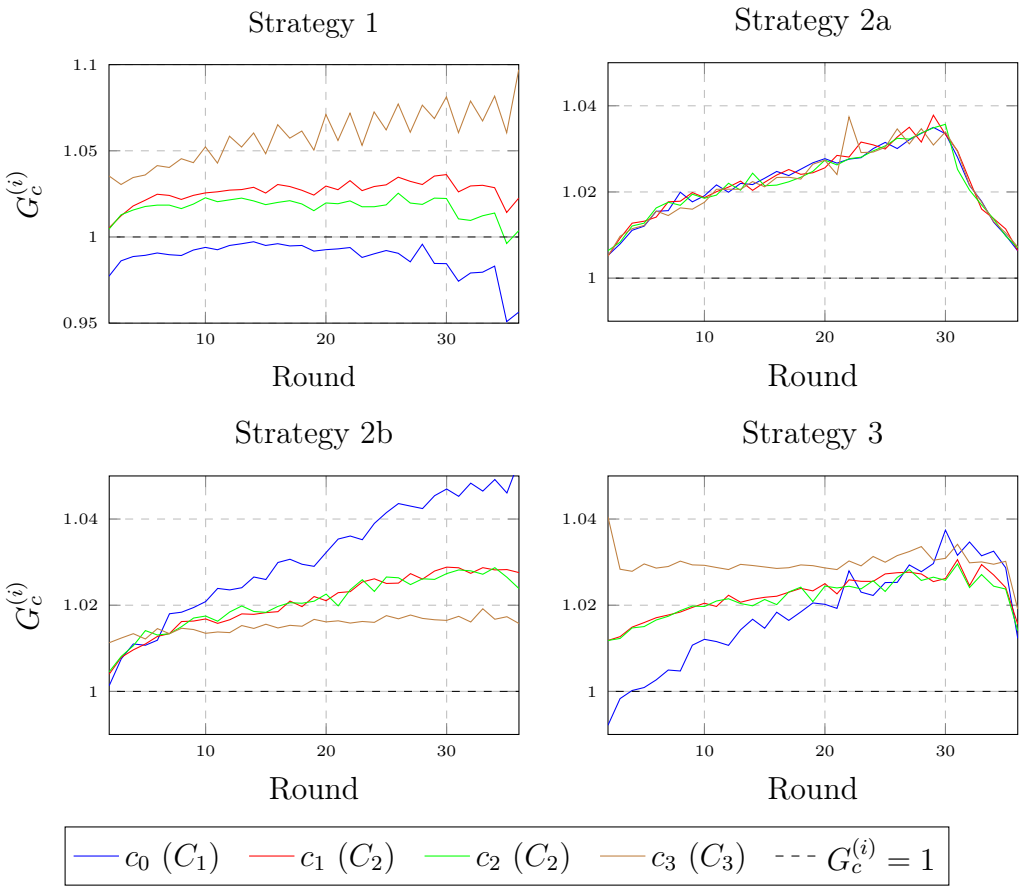


Figure 6.6: Client gap ratio G_c^i throughout training in each of the strategy. Note that the contribution of each of the clients varies depending on the choice of strategy. Similar G_c^i values indicate similar contribution of the clients to the aggregated model.

Dynamics of $G_c^{(i)}$ Explain the Behaviour of Strategies. From the optimisation perspective, strategy 2a is equivalent to a hardware-homogeneous setting due to gradient accumulation, which simulates larger mini-batch sizes. This is what I empirically confirm by comparing almost identical loss curve values and very similar values of $G_c^{(i)}$ between the clients. On the other hand, strategy 2b suggests that weaker clients performing smaller optimisation steps but in a greater number result in better, more similar models to the aggregated one. The growing discrepancy between the client gap ratios supports this hypothesis. The dynamics of $G_c^{(i)}$ in strategy 3, suggest an interesting, unintentional feature. Client contributions converging to the same value indicate that the adjusted number of local optimisation steps causes the models to become progressively similar in the sense of validation perplexity. Finally, note that different values of the client gap ratio between clients inhibit the immediate perplexity decay effect of the WSD learning rate scheduler. This may be a significant limitation of strategies 1 and 2b, explaining their worse performance.

6.2 Selective Multi-Head Attention

In this section, I start by developing an understanding of how the SMHA mechanism changes the training dynamics, and how it could be further applied to the federated scenario. Finally, I evaluate its performance in the hardware-heterogeneous case.

6.2.1 Centralised Study

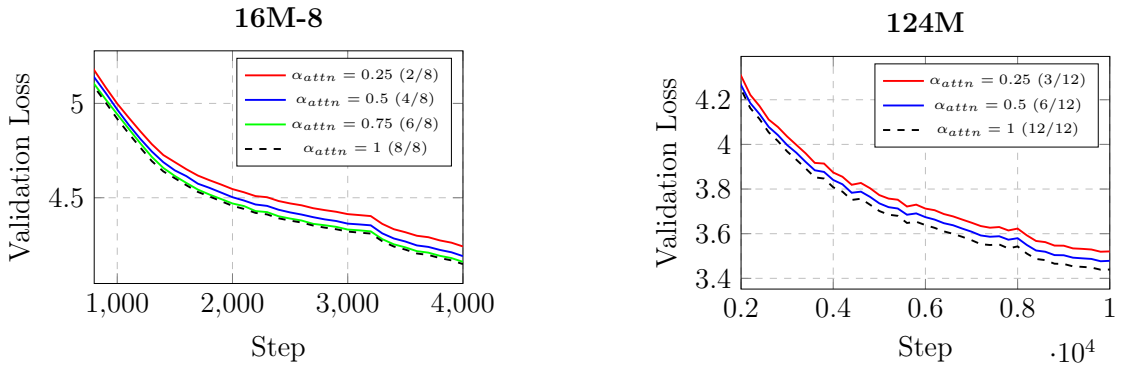


Figure 6.7: Validation loss comparison for 16M and 124M models with varying attention ratios. Excluding the first 20% of training for better readability. Results averaged over 3 seed runs.

Training Only Some Attention Heads Significantly Decreases Training Wall-Time but Only Marginally Worsens Perplexity. I start this analysis by comparing the performance of models trained with different values of $\alpha_{attn} < 1$, static throughout the duration of the training. As shown in Figure 6.7, training with only some of the available attention heads still leads to model convergence, and only slightly affects the final validation loss. Table 6.3 and Figure 6.8 show that while training with only a quarter of available heads worsens the final perplexity by about 2% for both model sizes, it results in almost 26% and 24% wall-time reduction for the smaller and larger models, respectively. A similar situation occurs for $\alpha_{attn} = 0.5$.

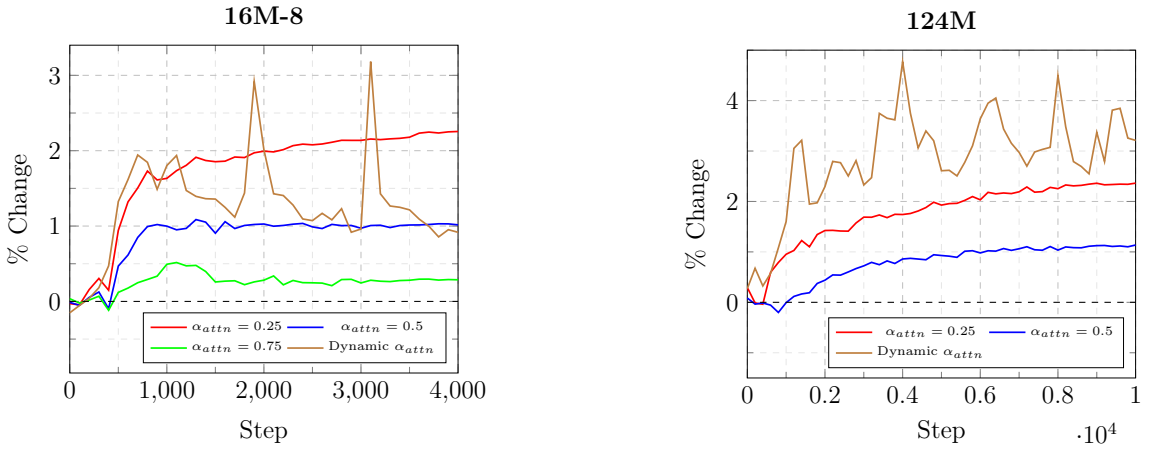


Figure 6.8: Change in the validation perplexity throughout model training. Results averaged over 3 seed runs apart from Dynamic α_{attn}.

	Training Time (±std)		% of $\alpha_{\text{attn}} = 1$ Training		FLOPs/it $\times 10^9$	
α_{attn}	**16M-8**	**124M**	**16M-8**	**124M**	**16M-8**	**124M**
0.25	1,145±0.20s	10,892s	74.16%	78.11%	0.098	0.802
0.5	1,190±0.62s	11,801s	77.07%	84.63%	0.100	0.859
0.75	1,283±9.04s	N/A	83.11%	–	0.101	–
1	1,544±7.68s	13,945s	100%	100%	0.103	0.973
Dynamic	1,187s	11,968s	76.89%	85.82%	–	–

Table 6.3: Influence of the attention head ratio α_{attn} on the training time and FLOPs per iteration. Results averaged over 3 seed runs apart from Dynamic α_{attn}.

Figure 6.8 shows the relative differences between the validation curves of runs with fractional α_{attn} and the full model. Interestingly, for both model sizes, the change in perplexity for static α_{attn} seems to be linearly scaled with the value of α_{attn}. This phenomenon means that the resulting lower perplexity comes from the insufficient expressive power of the models with limited attention mechanisms. Hence, it is reasonable to consider dynamically increasing the number of trainable heads. This would allow for a gradual increase in the capacity of the model, yet saving on training wall-time in the initial training phase. Finally, when applying SMHA to heterogeneous-hardware FL, I expect a decrease in wall-time, combined with a slightly worse validation perplexity.

Layer Attention Head Redundancy Decreases With Transformer Depth. Exploring how head redundancy heatmaps change over time hints that attention heads in different layers of the transformer learn to capture distinct patterns at different points during training. Figure 6.9 presents heatmaps from different stages of training a 124M model with all attention heads (iterations 400, 3,000 and 7,600 out of 10,000). In the beginning, most of the heads produce similar attention maps, hence, mean head redundancies are high. While the training progresses, all heads become more redundant, learning distinct patterns. Note, however, that in general, deeper layers of the transformer (7-11) have heads with lower redundancy scores than the early ones (0-3). Furthermore, heads in the first layer (0) seem to be capturing very similar patterns throughout the entirety of the training, only very gradually diverging towards the end.

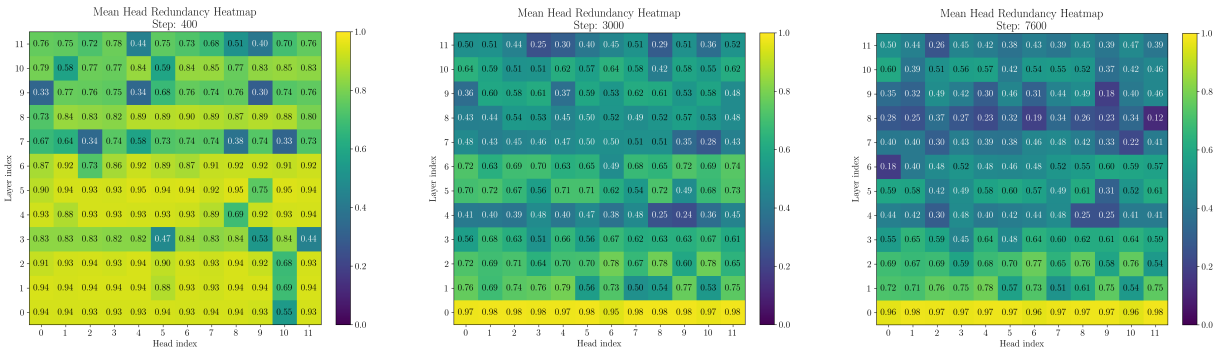
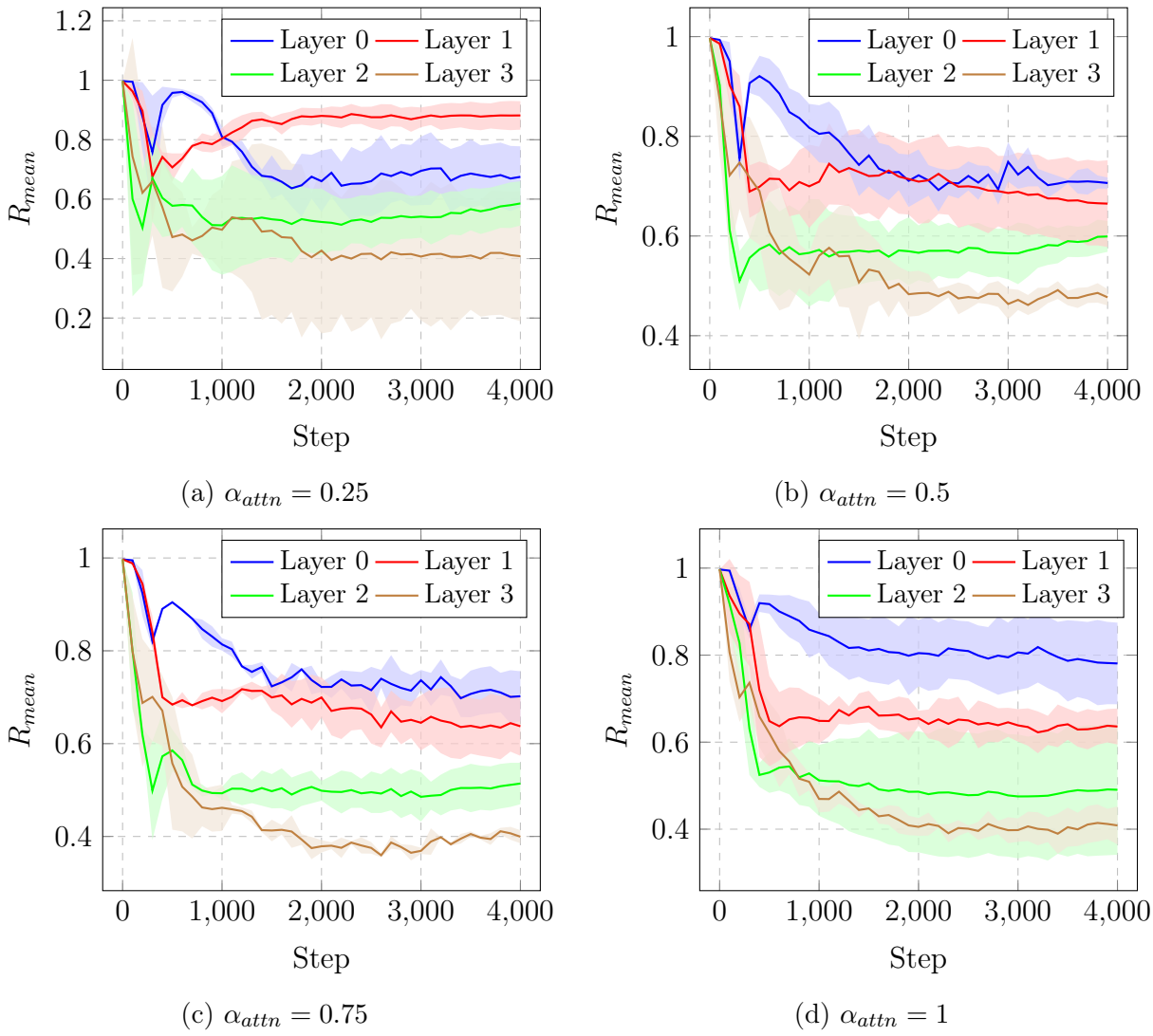


Figure 6.9: Head Redundancy Heatmaps for full-model training of a 124M model. Note the progressive reduction in overall redundancy apart from the first layer of the model.



(a) $\alpha_{attn} = 0.25$

(b) $\alpha_{attn} = 0.5$

(c) $\alpha_{attn} = 0.75$

(d) $\alpha_{attn} = 1$

Figure 6.10: Mean per-layer redundancy R_{mean} indicating that deeper layers learn more distinct patterns than the early ones. Results averaged over 3 seed runs.

Each layer of the model can be summarised using the layer mean redundancy metric R_{mean}. In Figures 6.10, I show how this metric changes over time for 16M-8 model training runs with various α_{attn}. Note that for all studied attention head ratios, the early layers (0, 1) display larger values of R_{mean}, and the deeper layers (2, 3) have generally less similar heads. Such a difference between layers can be explained assuming the common consensus that the earlier transformer layers capture the syntactic patterns, while the latter focus on the semantics, which are more numerous and distinct. Finally, it is worth noting that decreasing α_{attn} does not necessarily lead to lower head redundancy. This indicates that equipping models with a smaller number of attention heads does not necessarily encourage learning more distinct representations.

Layer Attention Head Redundancy Values Converge to Constant Levels. Results in Figures 6.10 show that after the brief initialisation period, the per-layer values of R_{mean} remain roughly constant. This suggests the existence of only a finite number of distinct patterns in the training data, which the heads are learning. It can also be observed

for the larger, 124M parameter model on the left in Figure 6.11, where all layers converge to some constant values. Notably, the first layer of the transformer displays particularly high mean redundancy, shown as the top light-blue curve close to 1. This confirms the observation of the corresponding head redundancy heatmaps in Figure 6.9, where heads in the first layer were very redundant. Furthermore, studying the maximum redundancy R_{max} shows a similar convergence pattern. This means that in every layer there exists a pair of heads, which capture relatively similar patterns ($R_{max} > 0.8$ for most of the layers). Since the occurrence of the convergence of R_{mean} and R_{max} is invariant of α_{attn}, it suggests that reducing the number of trained heads does not fundamentally change the properties of the models, only resulting in adequately scaled value of the validation loss.

However, the observed convergence of the redundancy metrics prompts their further study, since there is no evidence on how their values actually relate to the model performance. The model performance likely scales non-linearly with the values of redundancy, and hence, the change in model behaviour should be studied for an entire spectrum of them.

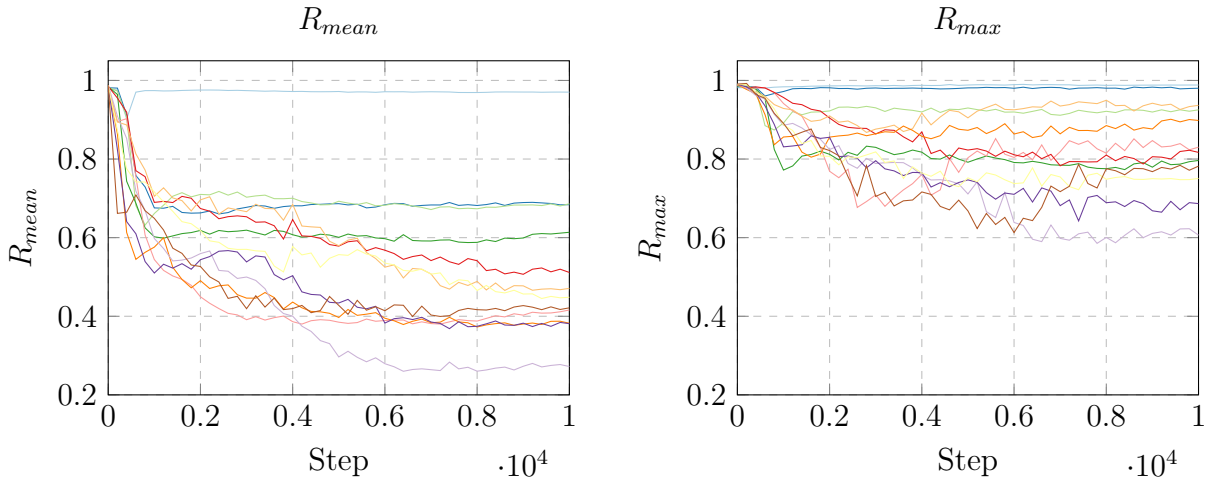


Figure 6.11: R_{mean} and R_{max} plots for a full-model training of a 124M model. Note, that every layers converges to some constant R_{mean}, while in every layer there exist a head with relatively high redundancy, measured through R_{max}.

Dynamically Adding Attention Heads Does Not Outperform Static α_{attn}. Implementing the idea to gradually increase the pattern-capturing capacity of the model does not necessarily lead to better results, compared with a static α_{attn}. Note, that in the case of 16M-8, the perplexity of the dynamic run is similar a static $\alpha_{attn} = 0.5$, with a marginally shorter training wall-time (left plot in Figure 6.8). However, the difference in validation perplexity compared to the baseline for the 16M-8 model is decreasing in the dynamic approach. Assuming the prior reasoning about the validation loss differences is correct, this suggests that dynamically increasing the number of trained heads indeed increases model complexity, allowing it to capture more patterns. On the other hand, this does not hold in the case of the larger model, where the relative change in perplexity to the baseline oscillates. Furthermore, the final validation loss is significantly worse than any studied cases with static α_{attn}.

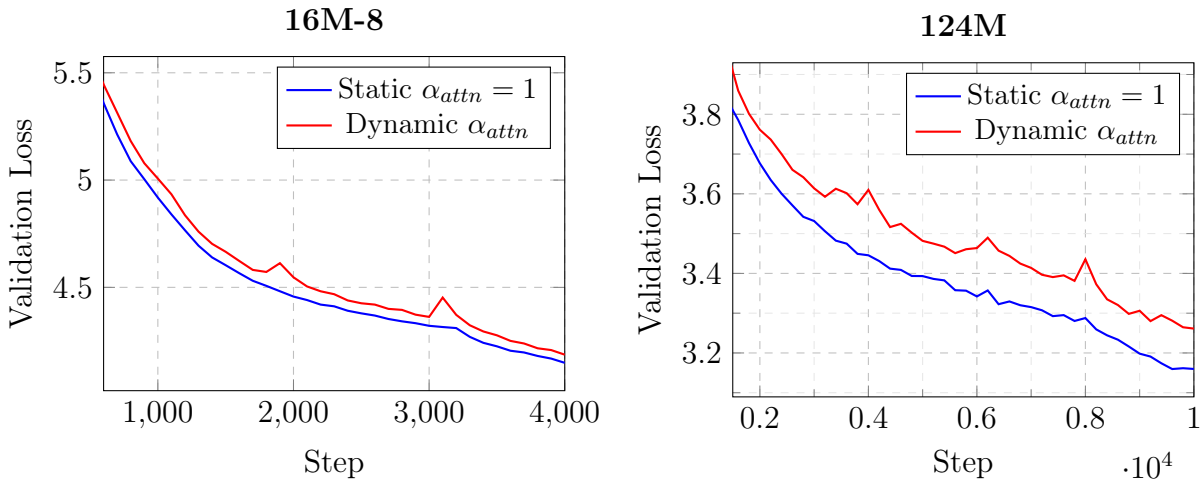


Figure 6.12: Influence of the dynamically varied α_{attn} on the validation loss dynamics. Note the spikes caused by head addition.

The issue with non-decreasing validation loss in the case of the larger model may be related to the observed spikes in the loss, likely caused by the addition of new untrained attention heads (Figure 6.12). In the case of the smaller model, the addition of heads occurred only a few times. This can be easily observed as immediate changes in the value of R_{max} for the layer which is resized (Figure 6.13). Note that these points coincide with the spikes in validation loss, supporting this hypothesis. Since the larger model involved significantly more changes in its architecture (Figure 6.14), it is likely that they significantly disturbed the validation loss, preventing better convergence. I conclude that the worse performance of the larger model could come from a poorly tuned scheduler for θ_R.

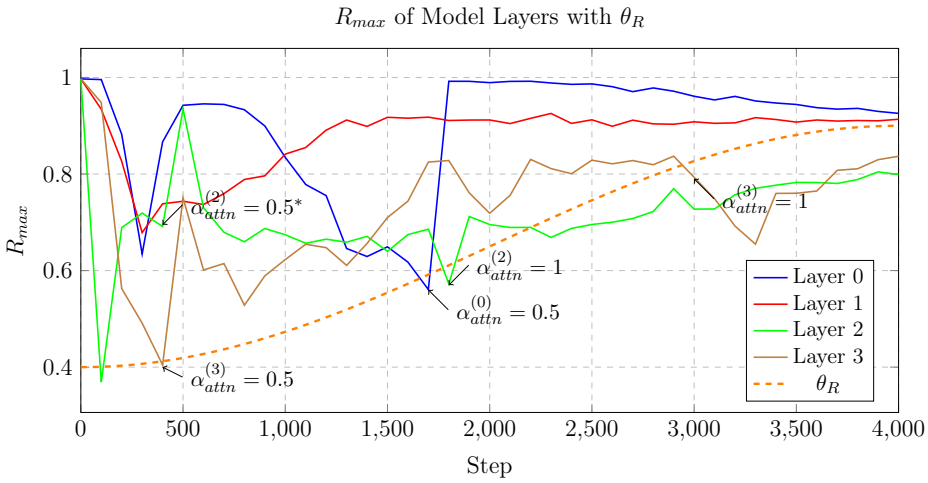


Figure 6.13: Dynamically increasing the attention ratio α_{attn} in each layer based on their corresponding R_{max} values and the threshold θ_R for the 16-8M model. Note the immediate increase, followed by a gradual decrease in redundancy after head addition (marked by arrows), caused by the initialisation. *While the R_{max} for layer 2 crossed the threshold earlier at iteration 100, it was in the 10% warmup period (400 iterations). Hence, the change of $\alpha^{(2)}_{attn}$ is delayed.

Note that increasing the number of heads per layer should increase the attention head

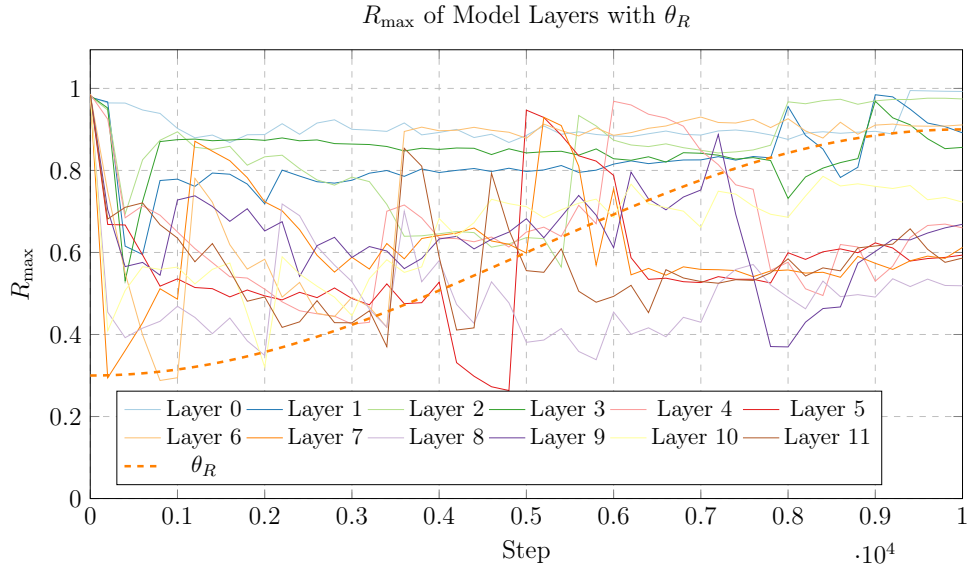


Figure 6.14: Dynamically increasing the attention ratio α_{attn} in each layer based on their corresponding R_{max} values and the threshold θ_R for the 124M model. The R_{max} of some layers decreases past the threshold in the second half of the training due to the maximum α_{attn} reached.

redundancy within it, since the added untrained heads produce similar representations. This behaviour is observed for both models in Figures 6.13 and 6.14. However, while the resulting large value of R_{max} decreases in some layers in the following optimisation steps, other layers exhibit the opposite behaviour, increasing their value. This is surprising and needs to be further understood.

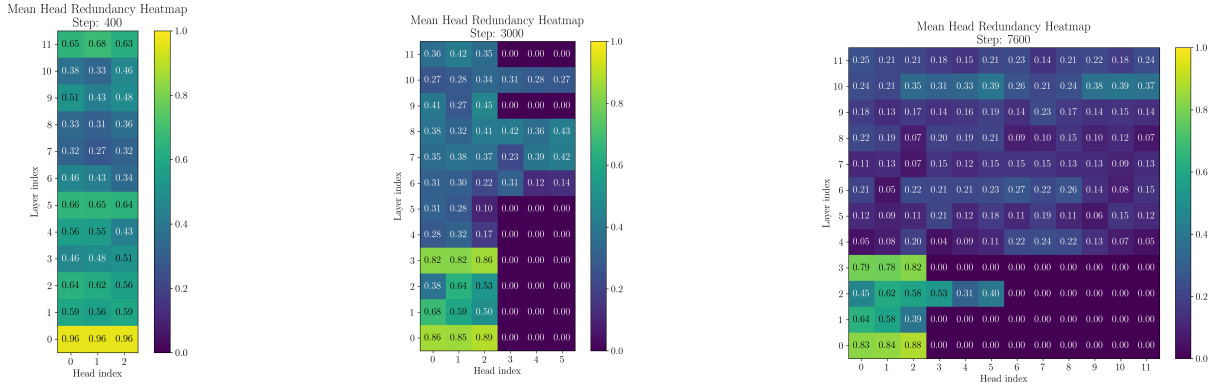


Figure 6.15: Head Redundancy Heatmaps for dynamically varied α_{attn}, individually for each layer of a 124M model. Note, that if a layer uses less heads than other layers, their entries are right-padded with 0s.

Dynamically adding attention heads changes the values observed in head redundancy heatmaps. Figure 6.15 shows that initialising the training with a significantly more constrained model leads to a faster decrease in head redundancy in every layer. While the shallow-deep split between the layers still holds, the head redundancy values in the early layers are much lower than when the full model is trained. This could be leveraged in model training as a faster initialisation method, followed by full or nearly full model

training. Finally, note how the iteration time increases when the new attention heads are added to the model (Figure 6.16). This displays that SMHA indeed shortens training time, and the per-iteration times can be granularly adjusted.

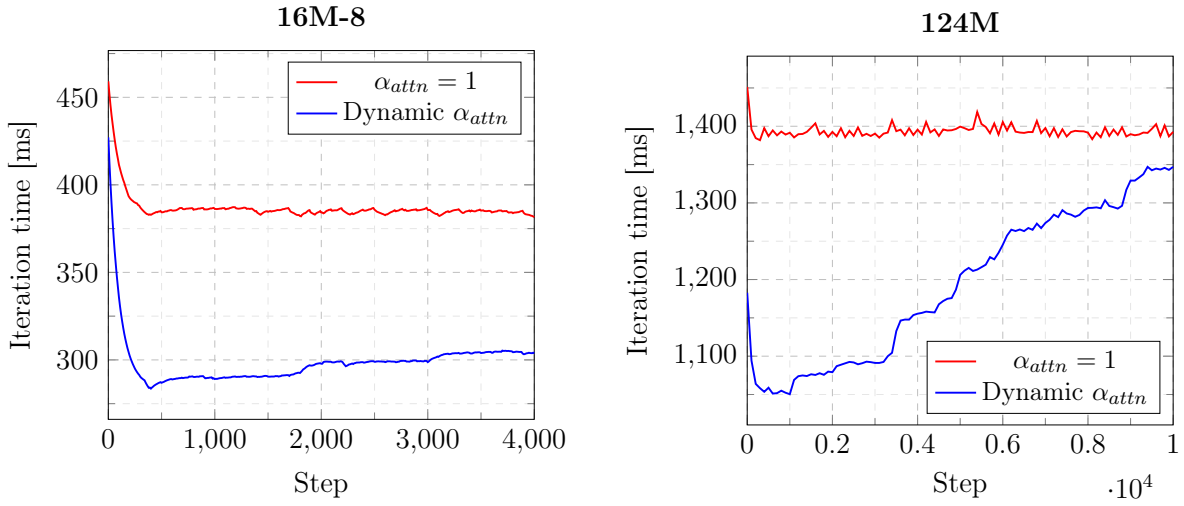
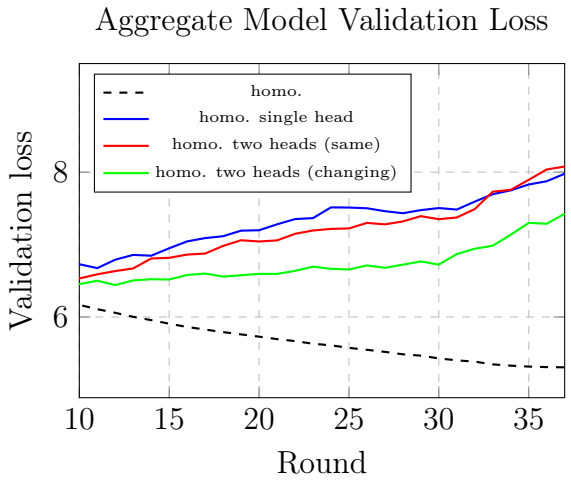


Figure 6.16: Difference in iteration times with dynamically varied α_{attn}. Note the progressive increase in iteration time due to more computation.

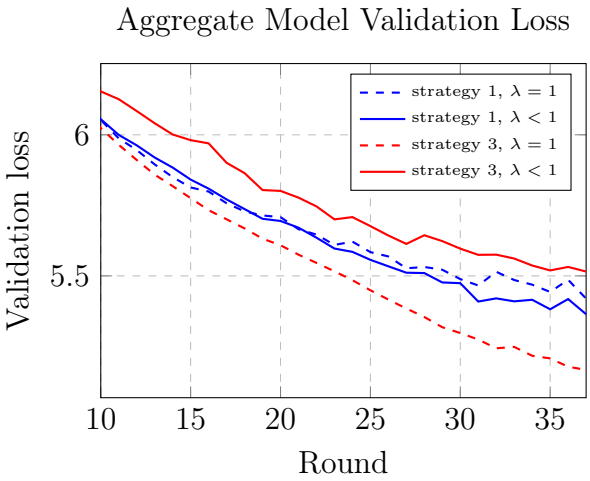
SMHA is Invariant to Scheduler Type but Warmup-Stable-Decay Achieves Lower Loss in General. Given a growing use of warmup-stable-decay schedulers in a single-epoch training regime, I studied whether the choice of the scheduler influences the performance of SMHA. It is particularly important from the perspective of dynamically changing α_{attn}, since annealing the learning rate may result in undertrained heads. Figure B.2 in Appendix B suggests that the WSD scheduler leads to a better performance. However, I compare it against the full model run, which displays that the WSD scheduler results in lower loss values in general (Figure B.1 in Appendix B). This means that either scheduler can be safely used with SMHA.

Chosen Scheme for Initialising Added Heads Does not Influence Model Performance. Comparing the initialisation methods shows that initialising new heads with the values of the already trained ones produces slightly larger loss spikes but the validation loss curves do not diverge, in general (Figure B.3 in Appendix B). Hence, to avoid copying time, I decide to initialise heads using a random initialisation scheme.

6.2.2 SMHA in Hardware-Heterogeneous FL



(a) In all studied cases, lack of at least one client with $\alpha_{attn} = 1$, leads to divergence of the aggregate model.



(b) Training with fractional λ results in similar perplexity when combined with strategy 1 but loss divergences compared to strategy 3.

Figure 6.17: Per-round validation perplexity of the aggregate 16M-4 model with SMHA. Training Each Attention Head Exactly Once Results in Divergence. I start the evaluation of SMHA in the federated setting by exploring the extreme case, where each attention head is trained only once by a corresponding client (1:1 mapping, $\lambda = 0.25$, $\alpha_{attn} = 0.25$). Training in a homogeneous-hardware setting significantly decreases training wall-time (84.67% of the full model training), as shown in Table 6.4. Nevertheless, averaging over only partially trained model parameters leads to divergence of the aggregated model (Figure 6.17a). On the other hand, local optimisation on the clients proceeds successfully, still minimising loss in every round. This can be observed by interpreting the decreasing client gap ratio below 1 (left diagram in Figure 6.18). Such a result suggests that the chosen aggregation strategy – FedAvg – may not be optimal in this setting. Hence, a different aggregation strategy may still exist, resulting in the convergence of the joint model.

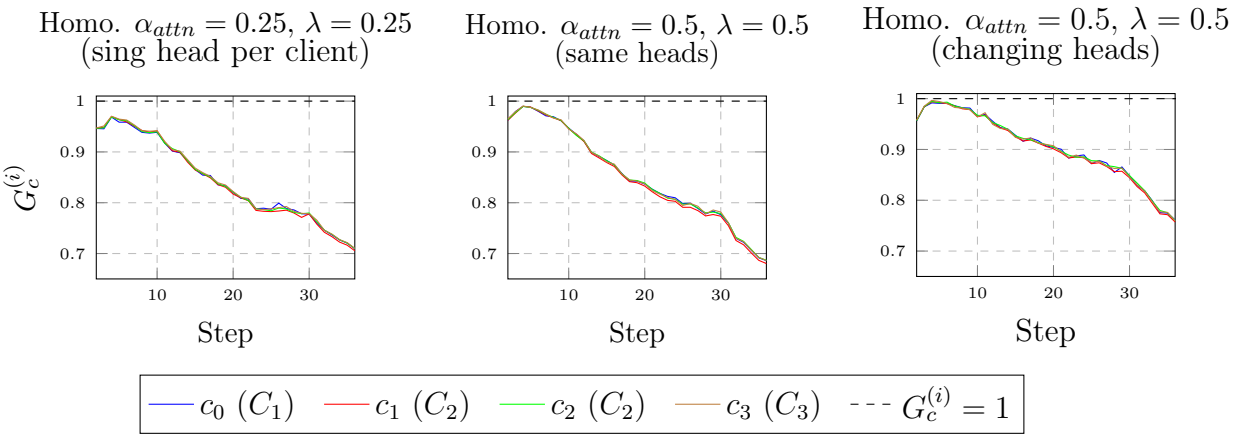


Figure 6.18: Client gap ratio $G_c^{(i)}$ for experiments with all clients $\alpha_{attn} < 1$.

Modifying Which Heads Are Trained Together Every Round Improves Model Performance. Doubling λ, and training two heads per client ($\alpha_{attn} = 0.5$) does not mitigate the issue of divergence, and leads to a similarly bad perplexity score (Figure 6.17a). However, there is a notable difference in validation loss curves between when clients always train the *same* pre-defined heads, and when the head assignments are *changing* every round (as described in Section 5.2.3). This means that when clients implement SMHA with $\alpha_{attn} < 1$, they should exchange the head assignments and train different components together. Finally, both homogeneous-hardware models trained with two heads per client complete in around 87% of the full model training wall-time, as expected from the centralised study.

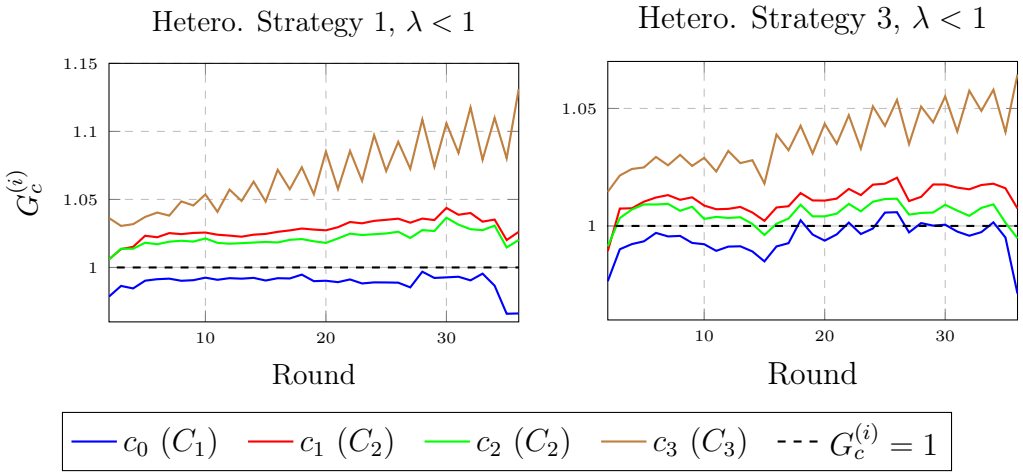


Figure 6.19: Client gap ratio $G_c^{(i)}$ for experiments applying SMHA to hardware-heterogeneity resolution strategies.

Convergence is Achieved When At Least One Client Trains All Heads. Modifying the federation, so that at least one client trains a complete model, leads to convergence (hetero. strategy 1 and hetero strategy 3, as described in Section 5.2.3). Note, however, that this may be a result of one client dominating the joint optimisation. This can be further supported by the differences in client gap ratio for strategy 1, as shown on the left in Figure 6.19. The client training the full model (c_0) consistently produces a better model than the aggregation of all clients, shown as $G_0^{(i)} < 1$ for all rounds i. Furthermore, the weakest client c_3 training just a single head, progressively diverges from the averaged model. On the other hand, strategy 3 results in more similar client gap ratios across, but still may have an issue of a dominating client.

Strategy 3 with SMHA Shortens Heterogeneous-Hardware Training Time But Increases Perplexity. Applying SMHA to hardware-heterogeneous configuration 2 with strategy 3, requires updating the t_{bmax} values for each client. The per-client mini-batch processing times and the corresponding α_{attn} are presented in Table 6.5. The decrease in mini-batch processing times achieved by applying SMHA allows for equalising more the share of samples, each client processes per round (Table 6.6. This translates into a 5% decrease in total training wall-time between strategy 3 with and without SMHA.

Strategy	Wall-Time (% baseline)	PPL (% baseline)
homo. $\alpha_{attn} = 0.25$, $\lambda = 0.25$	1,116s (84.67%)	7.9785 (150.41%)
homo. $\alpha_{attn} = 0.5$, $\lambda = 0.5$ (same heads)	1,150s (87.25%)	8.0779 (152.29%)
homo. $\alpha_{attn} = 0.5$, $\lambda = 0.5$ (changing heads)	1,148s (87.10%)	7.4250 (139.98%)
strategy 1, $\lambda < 1$	2,097s (100.96%)	5.364 (98.98%)
strategy 3, $\lambda < 1$	1,798s (95.85%)	5.515 (106.74%)

Table 6.4: Wall-time and perplexity changes over baselines. Note the divergence of runs where no client trains all attention heads.

Interestingly, this number is close to the maximum theoretical decrease (6.03%), which can be estimated by computing the average per-client decrease of t_{bmax} per iteration. Nevertheless, the SMHA-equipped configuration 2 with strategy 3 achieves 6.74% worse perplexity than the baseline without SMHA, with progressively diverging validation loss curves (Figure 6.17b). This result indicates that while shortening the wall-time, SMHA slows down the convergence of the aggregate model. While this study is limited in the number of considered client configurations and considers only a few client head assignments, larger federations may benefit from SMHA both in terms of wall-time and perplexity. Furthermore, a different aggregation strategy that is better suited for SMHA could result in better model convergence.

	16M			
Client Type	α_{attn}	$B_{\max}$	$t_{b_{\max}}$	Change in t_{bmax} compared to $\alpha_{attn} = 1$
C_1	1	32	0.165s	0%
C_2	0.5	16	0.120s	-7.0%
C_3	0.25	8	0.100s	-10.1%

Table 6.5: Applying SMHA to the hardware-heterogeneous four-client configuration decreases the mini-batch processing times.

c_i	Client type	Local steps L	% round samples with and without SMHA
		Strategy 3 – total samples/round: 16,384	
0	C_1	184	37.70% → 35.94%
1	C_2	253	24.22% → 24.71%
2	C_2	252	24.22% → 24.61%
3	C_3	302	13.87% → 14.75%

Table 6.6: Revised local step values for the 16M-4 model with SMHA.

Chapter 7

Conclusions and Future Work

The key findings of this dissertation can be grouped into four main areas. First, I showed that standard learning-rate tuning in a centralised setting does not carry over straightforwardly to federated averaging: across a wide grid of learning-rate values, the federated runs diverged from both small- and large-batch centralised baselines early in training, with final validation perplexities that tracked neither regime. Furthermore, synchronisation introduced by FedAvg fundamentally alters gradient-noise dynamics compared to centralised training. Instead of the steady noise-scale growth seen in centralised optimisation, federated clients' noise scale oscillates around a much lower constant value. This area is particularly exciting to study further since the "resets" of gradient noise after each round may enable the use of smaller local mini-batch sizes or indicate the need for a better compute-optimal mini-batch size criterion.

Second, in the presence of hardware heterogeneity, I evaluated four strategies for balancing local step counts and batch sizes. Strategy 3, which solves a small MILP to minimise the maximum idle time per round, achieves near-optimal wall-time utilisation, reducing system idle time several times compared to other schemes. Furthermore, evaluating the individual contributions of clients to the aggregate model in different strategies allows identification of participants, which are dominating the joint optimisation effort.

Third, introducing Selective Multi-Head Attention (SMHA) shows that statically training only a fraction of the model's attention heads yields a roughly linear degradation in perplexity, while reducing per-iteration compute. On the other hand, dynamically growing the attention head ratio based on per-layer redundancy metrics can further shorten training by gradually increasing the language modelling capacity of the models. Nevertheless, care must be taken when choosing at which points the head addition occurs, to avoid a gradual decrease in validation loss for larger models.

Finally, applying SMHA in the hardware-heterogeneous setting, so that each client trains only a fraction of the model's attention heads, yields a clear trade-off. By reducing per-client mini-batch processing times, SMHA better balanced workload across silos and

shortened overall wall-time. However, this efficiency gain came at the cost of slower convergence – the SMHA-equipped federation incurred worse validation perplexity compared to the full-model baseline.

Nevertheless, this work has several limitations. All experiments rely on a small number of clients, using solely the FedAvg aggregation strategy with a fixed number of communication rounds, without exploring other, possibly more robust optimisers such as FedAdam or FedYogi (Reddi et al., 2021). Furthermore, the study was conducted on relatively small GPT-style models (16M and 124M parameters) due to available computing resources. It may not directly translate to the hundreds-of-billions-parameter regime of state-of-the-art LLMs. Moreover, the SMHA mechanism depends on heuristic choices, such as the redundancy threshold schedule, cooldown periods, and warmup-stable-decay learning-rate policy, all of which may require extensive tuning for different architectures or training budgets.

Nevertheless, there are several promising directions for future work. One avenue is to extend the heterogeneity-resolution framework to incorporate adaptive aggregation algorithms that can better leverage stale or sparse updates, such as FedProx or FedAdam. Gradient noise scale measurements could be integrated into an automated batch-size tuner for federated clients, dynamically balancing computation and convergence, an idea loosely suggested by (McCandlish et al., 2018). While the thorough analysis of centralised SMHA uncovered different levels of parameter redundancy across different layers, the same study could be performed for its application to the federated case. Finally, this study was based on a limited in size and simulated experimental FL setup, which does not represent real-world deployments. Hence, validating contributions of this work with real-world cross-silo clients – with variable network latencies, bandwidth constraints, and on very large language models – will be crucial to assess both their practical utility and environmental impact.

To summarise, this dissertation explored multiple methods of increasing the efficiency of federated learning with hardware-heterogeneous clients. Combining idle time minimisation with the optimisation of training and architecture hyperparameters resulted in a trade-off between the model performance and training duration. The analysis of the experiments involving various compute-saving mechanisms uncovered new exciting avenues for further, more thorough examination.

Bibliography

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *arXiv preprint*. ArXiv:2005.14165 [cs].

Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. 2020. Asynchronous Online Federated Learning for Edge Devices with Non-IID Data. *arXiv preprint*. ArXiv:1911.02134 [cs].

Ziheng Cheng and Margalit Glasgow. 2025. Convergence of distributed adaptive optimization with local updates. *Preprint*, arXiv:2409.13155.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. *arXiv preprint*. ArXiv:2204.02311 [cs].

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. 2024a. MoEUT: Mixture-of-Experts Universal Transformers. *arXiv preprint*. ArXiv:2405.16039 [cs].

Róbert Csordás, Piotr Piękos, Kazuki Irie, and Jürgen Schmidhuber. 2024b. SwitchHead: Accelerating Transformers with Mixture-of-Experts Attention. *arXiv preprint*. ArXiv:2312.07987 [cs].

Arthur Douillard, Yanislav Donchev, Keith Rush, Satyen Kale, Zachary Charles, Zachary Garrett, Gabriel Teston, Dave Lacey, Ross McIlroy, Jiajun Shen, Alexandre Ramé,

Arthur Szlam, Marc'Aurelio Ranzato, and Paul Barham. 2025. Streaming DiLoCo with overlapping communication: Towards a Distributed Free Lunch. *arXiv preprint*. ArXiv:2501.18512 [cs].

Arthur Douillard, Qixuan Feng, Andrei A. Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. 2024. DiLoCo: Distributed Low-Communication Training of Language Models. *arXiv preprint*. ArXiv:2311.08105 [cs].

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *arXiv preprint*. ArXiv:2101.03961 [cs].

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

Adriano Guastella, Lorenzo Sani, Alex Iacob, Alessio Mora, Paolo Bellavista, and Nicholas D. Lane. 2025. SparsyFed: Sparse Adaptive Federated Training. *arXiv preprint*. ArXiv:2504.05153 [cs] version: 1.

Qiaozhi He, Xiaomin Zhuang, and Zhihua Wu. 2024. Exploring Scaling Laws for Local SGD in Large Language Model Training. *arXiv preprint*. ArXiv:2409.13198.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training Compute-Optimal Large Language Models. *arXiv preprint*. ArXiv:2203.15556.

Rui Hu, Yanmin Gong, and Yuanxiong Guo. 2022. Federated Learning with Sparsified Model Perturbation: Improving Accuracy under Client-Level Differential Privacy. *arXiv preprint*. ArXiv:2202.07178 [cs].

Alex Iacob, Lorenzo Sani, Bill Marino, Preslav Aleksandrov, William F. Shen, and Nicholas Donald Lane. 2024. Worldwide Federated Training of Language Models. *arXiv preprint*. ArXiv:2405.14446.

Jae-young Jo and Sung-Hyon Myaeng. 2020. Roles and Utilization of Attention Heads in Transformer-based Neural Language Models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3404–3417, Online. Association for Computational Linguistics.

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. *arXiv preprint*. ArXiv:2001.08361.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. 2019. Similarity of Neural Network Representations Revisited. *arXiv preprint*. ArXiv:1905.00414 [cs].

Shuaipeng Li, Penghao Zhao, Hailin Zhang, Xingwu Sun, Hao Wu, Dian Jiao, Weiyan Wang, Chengjun Liu, Zheng Fang, Jinbao Xue, Yangyu Tao, Bin Cui, and Di Wang. 2024. Surge phenomenon in optimal learning rate and batch size scaling. *Preprint*, arXiv:2405.14578.

Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. 2020. Federated Optimization in Heterogeneous Networks. *arXiv preprint*. ArXiv:1812.06127 [cs].

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. *arXiv preprint*. ArXiv:1711.05101 [cs].

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. 2018. An Empirical Model of Large-Batch Training. *arXiv preprint*. ArXiv:1812.06162.

H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. 2023. Communication-Efficient Learning of Deep Networks from Decentralized Data. *arXiv preprint*. ArXiv:1602.05629.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving Language Understanding by Generative Pre-Training.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. *Preprint*, arXiv:1910.10683.

Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. 2021. Adaptive federated optimization. *Preprint*, arXiv:2003.00295.

Lorenzo Sani, Alex Iacob, Zeyu Cao, Royson Lee, Bill Marino, Yan Gao, Dongqi Cai, Zexi Li, Wanru Zhao, Xinchi Qiu, and Nicholas D. Lane. 2024a. Photon: Federated LLM Pre-Training. *arXiv preprint*. ArXiv:2411.02908.

Lorenzo Sani, Alex Iacob, Zeyu Cao, Bill Marino, Yan Gao, Tomas Paulik, Wanru Zhao, William F. Shen, Preslav Aleksandrov, Xinchi Qiu, and Nicholas D. Lane. 2024b. The Future of Large Language Model Pre-training is Federated. *arXiv preprint*. ArXiv:2405.10853.

Le Song, Alex Smola, Arthur Gretton, Karsten Borgwardt, and Justin Bedo. 2007. Supervised Feature Selection via Dependence Estimation. *arXiv preprint*. ArXiv:0704.2668 [cs].

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. *arXiv preprint*. ArXiv:1706.03762.

Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H. Vincent Poor. 2020. Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization.

Zhuofan Xia, Xuran Pan, Xuan Jin, Yuan He, Hui Xue, Shiji Song, and Gao Huang. 2023. BUDGETED TRAINING FOR VISION TRANSFORMER.

Rui Ye, Wenhao Wang, Jingyi Chai, Dihan Li, Zexi Li, Yinda Xu, Yaxin Du, Yanfeng Wang, and Siheng Chen. 2024. OpenFedLLM: Training Large Language Models on Decentralized Private Data via Federated Learning. *arXiv preprint*. ArXiv:2402.06954.

Binhang Yuan, Yongjun He, Jared Quincy Davis, Tianyi Zhang, Tri Dao, Beidi Chen, Percy Liang, Christopher Re, and Ce Zhang. 2023. Decentralized Training of Foundation Models in Heterogeneous Environments. *arXiv preprint*. ArXiv:2206.01288.

Xiaofeng Zhang, Yikang Shen, Zeyu Huang, Jie Zhou, Wenge Rong, and Zhang Xiong.

2022. Mixture of Attention Heads: Selecting Attention Heads Per Token. *arXiv preprint*. ArXiv:2210.05144 [cs].

Appendix A

Code Listing

A.1 Centralised Experiments

The experimental code has been initialised with the *nanoGPT* project. The repository includes heavily modified training loop and new definitions of SMHA-equipped GPT models. Furthermore, it includes software for tracking and analysing attention head redundancy, as well as the script solving the *strategy 3* optimisation problem. The exact contributed code will be publicly available soon.

A.2 Federated Learning Experiments & Other changes

The development of the experimental setup involved several changes and adaptations to existing codebases. The changes made in the duration of this project will be published soon.

Appendix B

SMHA Ablation Study Plots

B.1 Choice of Scheduler and Model Convergence

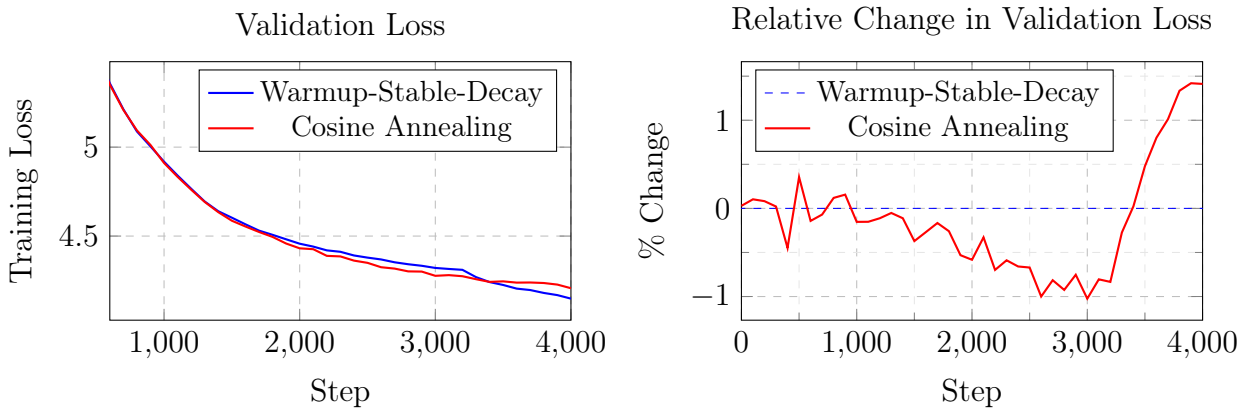


Figure B.1: Results from training the full model with static $alpha_{attn} = 1$. **Left:** Validation losses of the WSD and Cosine Annealing scheduler. **Right:** Relative change in Validation loss between Cosine Annealing and WSD schedulers.

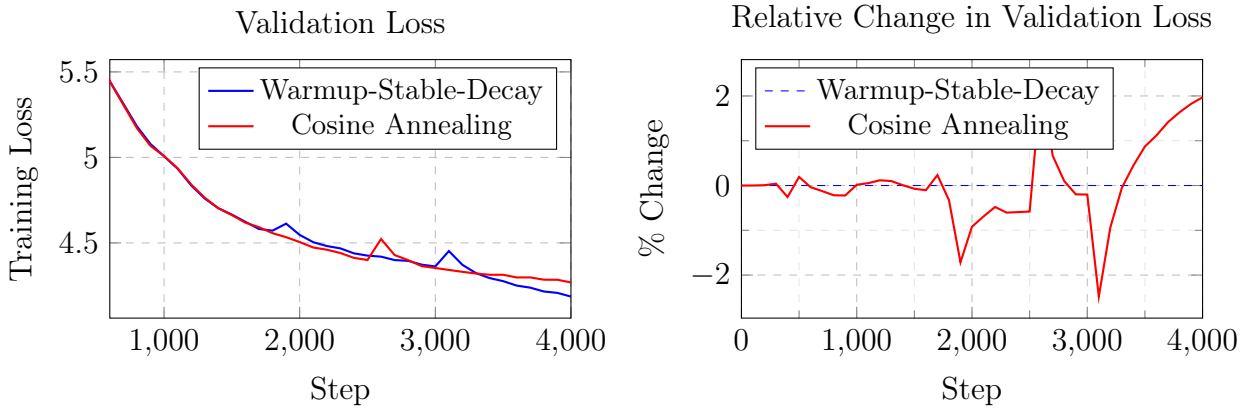


Figure B.2: Results from dynamically scheduled $alpha_{attn}$. **Left:** Validation losses of the WSD and Cosine Annealing scheduler. **Right:** Relative change in Validation loss between Cosine Annealing and WSD schedulers.

B.2 Attention Head Initialisation Method

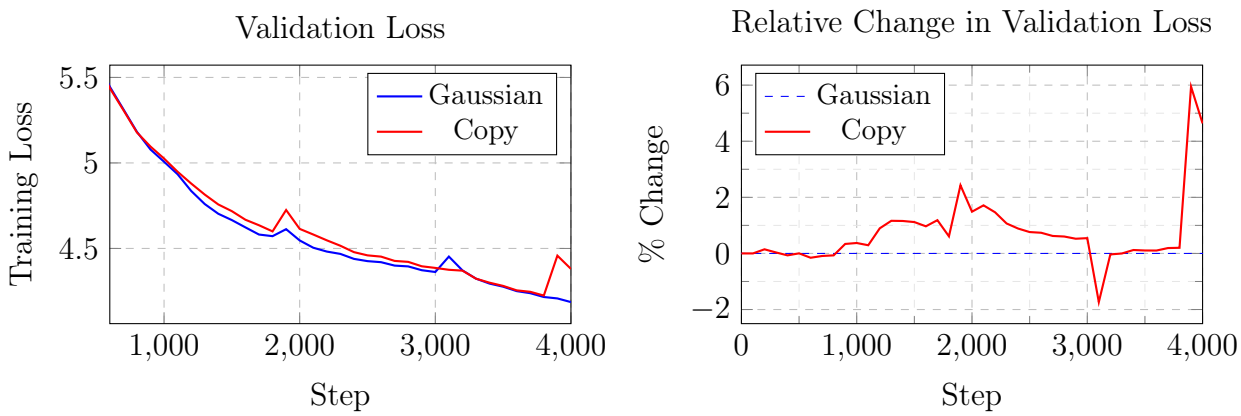


Figure B.3: Influence of the attention head initialisation scheme. **Left:** Validation losses of the Gaussian and Copy schemes. **Right:** Relative change in Validation loss between Copy and Gaussian schemes.