

Scaling Laws for Compute Optimal Biosignal Transformers

by

Thomas Fortin

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2024

© Thomas Fortin 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Scaling laws which predict the optimal balance between number of model parameters and number of training tokens given a fixed compute budget have recently been developed for language transformers. These allow model developers to allocate their compute budgets such that they can achieve optimal performance. This thesis develops such scaling laws for the Biosignal Transformer trained separately on both accelerometer data and EEG data. This is done by applying methods used by other researchers to develop similar scaling laws for language transformer models. These are referred to as the iso-FLOP curve method and the parametric loss function method.

The Biosignal Transformer model is a transformer model which is designed specifically to be trained on tasks that use biosignals such as EEG, ECG, and accelerometer data as input. For example, the Biosignal Transformer can be trained to detect or classify seizures from EEG signals. The Biosignal Transformer is also of particular interest because it is designed to use unsupervised pre-training on large unlabeled biosignal datasets to improve performance on downstream tasks with smaller labeled fine-tuning datasets. This work develops scaling laws which optimize for the best unsupervised pre-training loss given a fixed compute budget.

Results show that the developed scaling laws are successful at predicting a balance between number of parameters and number of training tokens for compute budgets five times larger than those used to develop them such that pre-training loss is minimized. Researchers who intend to scale up the Biosignal Transformer should use these scaling laws to attain optimal pre-training loss from their given compute budgets when applying unsupervised pre-training with the Biosignal Transformer.

Acknowledgments

This research was sponsored by Smile Digital Health.

I'd also like to thank my supervisor, Bryan Tripp, for his advice and guidance in my research.

Finally, I'd like to thank Cathleen for all the love and support she has provided me throughout my degree and research.

Table of Contents

Author’s Declaration	ii
Abstract	iii
Acknowledgments	iv
List of Figures	ix
List of Tables	xi
List of Abbreviations	xiv
1 Introduction	1
1.1 Summary of Contributions	2
2 Background	5
2.1 Transformers	5
2.2 Language Transformers	9
2.3 BIOT	10
2.4 Maximal Update Parametrization	12
2.5 Training Compute Optimal Large Language Models	14
2.6 Scaling Laws for Neural Language Models	17

3	Experiment Setup	20
3.1	Datasets	20
3.1.1	National Health and Nutrition Examination Survey	20
3.1.2	Temple University Hospital EEG	21
3.1.3	Data loading	22
3.2	Maximal Update Parametrization for BIOT	22
3.2.1	Implementation	23
3.2.2	Coordinate checks	24
3.2.3	Upscaling verification	25
3.2.4	Deficiencies at small sizes	26
3.3	Hyperparameter Selection	27
3.3.1	Shape	27
3.3.2	Preprocessing related	28
3.3.3	Batch size	29
3.3.4	Optimizer related	29
3.4	Insights into Overfitting	30
3.4.1	Tests	31
3.4.2	Results	31
3.4.3	Discussion	32
4	Scaling Law Development	37
4.1	Accelerometer–Iso-FLOP Method Attempt 1	38
4.1.1	Training runs	39
4.1.2	Iso-FLOP curves	40
4.1.3	Scaling law testing	43
4.1.4	Discussion	43
4.2	Accelerometer–Iso-FLOP Method Attempt 2	45
4.2.1	Training runs	45

4.2.2	Iso-FLOP curves	46
4.2.3	Scaling law testing	48
4.2.4	Discussion	49
4.3	Accelerometer–Parametric Loss Function Method	49
4.3.1	Parametric loss function	50
4.3.2	Iso-FLOP curves	50
4.3.3	Scaling law testing	52
4.3.4	Discussion	52
4.4	EEG–Iso-FLOP Method	53
4.4.1	Training runs	54
4.4.2	Iso-FLOP curves	55
4.4.3	Scaling law testing	55
4.4.4	Discussion	56
4.5	EEG Parametric Loss Function Method	57
4.5.1	Parametric loss function	57
4.5.2	Iso-FLOP curves	57
4.5.3	Scaling law testing	59
4.5.4	Discussion	60
5	Conclusion	62
5.1	Limitations	63
5.2	Future Work	64
	References	66
	APPENDICES	72
A	TUH-EEG Channels	73
B	Additional EEG Analysis	75

C	Fine Tuning Analysis	77
C.1	Datasets	78
C.1.1	Temple University Hospital Abnormal	78
C.1.2	Temple University Hospital Events	78
C.2	Training and Testing Procedure	79
C.2.1	TUAB	80
C.2.2	TUEV	80
C.3	Constant Compute Budget–TUAB	80
C.4	Varying Compute Budget–TUAB	81
C.5	Constant Compute Budget TUEV	83
C.6	Varying Compute Budget–TUEV	85
C.7	Analysis and Discussion	86

List of Figures

2.1	Scaled dot product attention proposed by Vaswani <i>et al.</i> [37].	6
2.2	Multi-head attention.	7
2.3	Transformer architecture proposed by Vaswani <i>et al.</i>	8
3.1	Coordinate check for a correct maximal update parametrization implementation for BIOT.	24
3.2	Coordinate check for an incorrect maximal update parametrization implementation for BIOT.	25
3.3	An example of an upscaling verification test which indicates an incorrect implementation of maximal update parametrization.	34
3.4	The upscaling verification test done on the maximal update parametrization implementation that was eventually used for the scaling law development. .	35
3.5	Training loss curves for the smallest model sizes used	36
4.1	Iso-FLOP curves for NHANES [1] data using results from Section 4.1.1 with a wide range of compute budgets.	41
4.2	Iso-FLOP curves for NHANES [1] data using results from Section 4.1.1 with a reduced range of compute budgets.	42
4.3	Training curves to test the first iteration of the accelerometer scaling law. .	44
4.4	The iso-FLOP curves generated in Section 4.2.2	47
4.5	Training curves to test the accelerometer scaling law developed in Section 4.2	49
4.6	The iso-FLOP curves generated using the parametric loss function fitted on accelerometer training curve data from Section 4.2.1.	51

4.7	Training curves to test the accelerometer scaling law developed with the parametric loss function method.	53
4.8	The iso-FLOPs using the EEG training curve data.	56
4.9	Training curves to test the EEG scaling law developed with the iso-FLOP curve only approach.	58
4.10	The iso-FLOPs using the parametric loss function fitted on EEG training curve data.	59
4.11	Training curves to test the EEG scaling law developed with the parametric loss function approach.	61
B.1	Combined results of each EEG scaling law test from Tables 4.8 and 4.10. .	76

List of Tables

2.1	Results from tests with BIOT from [43] trained on EEG data.	13
2.2	Results from tests with BIOT from [43] trained on ECG and accelerometer data.	13
2.3	Maximal update parametrization (μP) and standard parametrization (SP) multipliers.	15
3.1	The results from the overfitting tests done with the NHANES [1] dataset. .	31
3.2	The results from the overfitting tests done with the TUH-EEG [28] dataset.	32
4.1	Breakdown of training runs done in Section 4.1.1.	39
4.2	The results from the test of the scaling law developed in Section 4.1.	43
4.3	The training runs done for Section 4.2.1.	46
4.4	The results from the test of the scaling law developed in Section 4.2.	48
4.5	Results from fitting equation 4.4 to the experimental training loss curves from Section 4.2.1	50
4.6	The results from the test of the scaling law developed in Section 4.3.	52
4.7	The training runs done for the development of a scaling law with EEG data.	55
4.8	The results from the test of the EEG scaling law developed with the iso-FLOP curve method.	57
4.9	Results from fitting equation 4.4 to the experimental training loss curves from Section 4.4.1	58
4.10	The results from the test of the EEG scaling law developed with the parametric loss function approach.	60

5.1	The resulting scaling laws from Chapter 4.	63
A.1	The 24 channels that were selected to be kept from the TUH-EEG [28] dataset.	74
C.1	Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 25 steps.	81
C.2	Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 100 steps.	81
C.3	Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 400 steps.	81
C.4	Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 1600 steps.	82
C.5	Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 25 steps.	82
C.6	Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 100 steps.	82
C.7	Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 400 steps.	83
C.8	Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 1600 steps.	83
C.9	Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the TUEV dataset with 25 steps.	84
C.10	Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the TUEV dataset with 100 steps.	84
C.11	Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the TUEV dataset with 400 steps.	84
C.12	Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the TUEV dataset with 1600 steps.	84
C.13	Comparison of mean Cohen’s Kappa score across models with varying com- pute budgets fine-tuned on the TUEV dataset with 25 steps.	85
C.14	Comparison of mean Cohen’s Kappa score across models with varying com- pute budgets fine-tuned on the TUEV dataset with 100 steps.	85

C.15 Comparison of mean Cohen’s Kappa score across models with varying compute budgets fine-tuned on the TUEV dataset with 400 steps.	85
C.16 Comparison of mean Cohen’s Kappa score across models with varying compute budgets fine-tuned on the TUEV dataset with 1600 steps.	86
C.17 T-tests comparing mean of optimal pre-trained models vs larger pre-trained models.	88

List of Abbreviations

- AUROC** area under the receiver operating characteristic curve 12, 13, 80–83, 86
- BERT** Bidirectional Encoder Representations from Transformers 2, 3, 9–11
- BIOT** Biosignal Transformer 1–3, 10–13, 19, 20, 22–30, 37, 38, 49, 52, 54, 57, 62–65, 77
- ECG** electrocardiogram 1, 11–13
- EEG** electroencephalogram 1–4, 10–13, 21, 25, 28, 29, 33, 37, 38, 53–64, 75, 76, 78, 80, 81, 83, 86
- FFCL** CNN-LSTM feature fusion network 12, 13
- FLOP** floating point operations 1, 3, 4, 16, 22, 30, 37–51, 53–60, 62–64, 75
- HAR** human activity recognition 1, 11–13
- NHANES** National Health and Nutrition Examination Survey 20, 22, 25, 28–36, 38, 39, 41, 42, 53–55, 57, 59, 60
- PTB-XL** Physikalisch Technische Bundesanstalt-XL 12, 13
- S3T** Spatial-Temporal Tiny Transformer 12, 13
- SHHS** Sleep Heart Health Study 11, 13
- TUAB** Temple University Abnormal 12, 13, 77–83, 87, 88
- TUEV** Temple University Events 12, 13, 77, 79, 80, 83–88
- TUH-EEG** Temple University Hospital EEG 21, 28–33, 53, 54, 57, 59, 65, 73–75, 77, 78, 86

Chapter 1

Introduction

Transformer models are a class of artificial neural networks which were initially proposed in 2017 [37] to be applied to language modeling tasks. In recent years, a class of language modeling transformers called large language models have led the way in significant performance improvements for language modeling tasks [35, 32, 9, 36, 4, 30]. These models achieve this high performance level through significant increases in model size and training dataset size. At these massive scales, the costs to train such a model are significant. This makes the efficient use of a compute budget of paramount importance. As such, research has been done to create scaling laws for large language models which predict the optimal balance of number of model parameters versus number of training tokens seen during training [20, 16]. These scaling laws usually take the form of power law relationships between the compute budget (measured in [floating point operations \(FLOP\)](#)) and each of number of tokens and number of parameters [20, 16].

While transformers have been leading the way in advances in language modeling, they are also more commonly being applied with success to other areas such as computer-vision [11, 12, 21] and signal processing [38, 15]. Of interest to this thesis is biosignal processing. The [Biosignal Transformer \(BIOT\)](#) [43] is a transformer model which can be applied to tasks such as seizure detection and classification with [electroencephalogram \(EEG\)](#) data, arrhythmia detection and classification with [electrocardiogram \(ECG\)](#) data, and [human activity recognition \(HAR\)](#) with accelerometer data.

BIOT is particularly of interest since it is designed to make use of unsupervised pre-training¹ on unlabeled datasets of biosignals before being fine-tuned on one of the previously listed tasks. This makes it a good candidate for upscaling since acquiring large amounts

¹Some may refer to this as self-supervised learning which is a subset of unsupervised learning. Yang *et*

of unlabeled data is less challenging than acquiring large datasets of labeled data. A first step towards scaling up BIOT is to develop a scaling law which predicts the best balance of number of model parameters and number of training tokens for larger compute budgets. In this thesis, I adopt the power law model of compute-optimal scaling laws and apply methods used by Hoffmann *et al.* [16] in their development of such a scaling law for language transformers to fit scaling laws for BIOT with both EEG and accelerometer data. This should be useful to researchers intending to improve performance on biosignal processing tasks by scaling up BIOT as well as those interested in developing foundational models for biosignal processing tasks using BIOT.

1.1 Summary of Contributions

The goal of this thesis is to develop a scaling law for BIOT [43] which predicts the balance of number of parameters and number of training tokens for a given compute budget which will minimize unsupervised pre-training loss with enough accuracy for it to be useful for other researchers who intend to scale up BIOT. I do this separately for BIOT pre-trained on accelerometer data and for BIOT pre-trained on EEG data. For each modality, I apply two of the three methods defined by Hoffmann *et al.* [16] in their development of a scaling law for language transformer models. I then test the resulting scaling laws similarly to how Hoffmann *et al.* [16] test theirs.

The result of this work is 2 scaling laws for unsupervised pre-training of BIOT on accelerometer data and 2 scaling laws for unsupervised pre-training of BIOT on EEG data. Researchers and developers who wish to scale up BIOT with larger unsupervised pre-training compute budgets can use these scaling laws to predict the best balance of number of parameters and number of training tokens to get the lowest pre-training loss from their efforts.

- In Chapter 2, an overview of relevant literature is provided.
 - Section 2.1 discusses the original work by Vaswani *et al.* [37] which first proposed the transformer architecture. Topics including the attention mechanism, multi-head attention, and encoder vs decoder architectures are discussed.
 - Section 2.2 discusses language transformers and specifically the work of Devlin *et al.* [10] in their development of the Bidirectional Encoder Representations

al. [43] refer to their method of pre-training on unlabeled biosignal data as “unsupervised pre-training”. To maintain consistency with their work, that same terminology is used here.

- from Transformers (BERT) model. Particular focus is given to methods for generating embeddings for text as well as the pre-training tasks used for this and other similar language transformer models.
- Section 2.3 discussed the BIOT [43] model architecture. The method it uses to generate embeddings for biosignals, the model’s architecture, as well as its chosen pre-training tasks are presented with some comparisons made to language transformers such as BERT [10].
 - Section 2.4 discusses maximal update parametrization [44]; a method for allowing hyperparameter transfer between models of different sizes.
 - Section 2.5 discusses the work of Hoffmann *et al.* [16] on developing a scaling law to predict the optimal number of model parameters and number of training tokens for language transformer models with a given compute budget. The two methods adapted from that work in this thesis are discussed in most detail.
 - Section 2.6 discusses the work of Kaplan *et al.* [20] in which they develop scaling laws for many aspects of neural language models including to model the optimal allocation of compute budget.
- In Chapter 3 the work done to prepare for scaling law development prior to the implementation of Hoffmann *et al.*’s [16] methods is documented.
 - Section 3.1 discusses the accelerometer and EEG datasets used for pre-training and evaluating BIOT.
 - Section 3.2 documents the implementation of maximal update parametrization [44] for BIOT.
 - Section 3.3 documents the selected hyperparameters and approaches used to select them.
 - Section 3.4 explores the potential for overfitting by repeating training data at different frequencies and discusses the potential presence of overfitting in the development of the scaling laws.
 - In Chapter 4 the two chosen methods from the work of Hoffmann *et al.* [16] are applied to BIOT [43] in order to develop a scaling law for each dataset using each method.
 - Section 4.1 documents an attempt at applying the iso-FLOP curve method to develop a scaling law with accelerometer data which results in a scaling law which is found to be inaccurate.

- Section 4.2 documents an attempt at applying the iso-FLOP curve method to develop a scaling law with accelerometer data again, with some modifications made based on observations from the first attempt, which results in an accurate scaling law.
 - Section 4.3 documents an attempt at applying the parametric loss function method to develop a scaling law with accelerometer data which results in a scaling law which is found to be accurate.
 - Section 4.4 documents an attempt at applying the iso-FLOP curve method to develop a scaling law with EEG data which results in a scaling law which is found to be accurate.
 - Section 4.5 documents an attempt at applying the parametric loss method to develop a scaling law with EEG data which results in a scaling law which is found to be accurate.
- In Chapter 5 the results of this thesis are summarized, conclusions and recommendations are made, and future work is discussed.

Chapter 2

Background

In this section, important background information for this thesis is presented. Each subsection effectively summarizes a relevant academic publication. The reader with existing experience in language transformers will likely be already familiar with the works presented in Sections 2.1 and 2.2 as they review the architectures of language transformers. However, Sections 2.3, 2.4, 2.5, and 2.6 may be new to most and are highly referenced throughout this thesis.

2.1 Transformers

Before transformers were introduced by Vaswani *et al.* [37] in 2017, convolutional and recurrent networks which used attention were state-of-the-art for sequence transduction tasks such as language translation [37]. Transformers were designed to use attention but to avoid both convolution and recurrence as these architectural designs lead to computational inefficiencies in training and inference [37]. Namely, recurrent neural networks are not very parallelizable. Avoiding recurrence can allow models to be fit more efficiently using available hardware. Convolutional neural networks require an increasing number of computations to apply attention to tokens which are further away from each other in a sequence. This issue is not present in transformers as the computational cost to apply attention between any two tokens is independent of distance within the sequence [37]. By avoiding these inefficiencies but still taking advantage of attention, transformers were shown to outperform other models (even ensemble models) significantly in machine translation tasks [37].

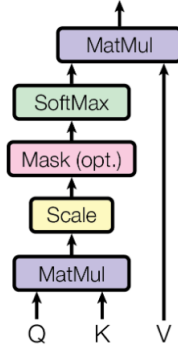


Figure 2.1: Scaled dot product attention proposed by Vaswani *et al.* [37]. This diagram includes a masking operation which can be used for masked language modeling as described next in Section 2.2. Adapted with permission from [37]

Attention involves query, key, and value vectors. These are linear projections of the attention mechanism’s input which are governed by learned projection matrices W_q , W_k , and W_v respectively. Query and key vectors have dimensionality of d_k and value vectors have dimensionality of d_v . Vaswani *et al.* [37] compute attention by first staking all query, key, and value vectors generated from a given sequence into matrices Q , K , and V . Then, the dot-product attention is given by:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad [37] \quad (2.1)$$

Vaswani *et al.* [37] note that scaling by $\frac{1}{\sqrt{d_k}}$ was a novelty in dot-product attention which allows it to perform well even as d_k becomes large. They theorized that since the variance (and average magnitude) of the dot-product of query and key vectors grows proportionally to d_k , the optimization of models with larger d_k will be hindered by exceedingly small gradients of the softmax function for large values. Scaling by $\frac{1}{\sqrt{d_k}}$ helps to mitigate this. A diagram representing this scaled dot-product attention is shown in Figure 2.1.

Although this method is able to overcome some of the drawbacks of convolutional and recurrent blocks, Vaswani *et al.* [37] identify a new weakness. The heavy averaging in the attention mechanism reduces effective resolution. To solve this, multi-head attention was introduced [37]. In somewhat similar fashion to convolutional neural networks with many kernels, multi-head attention applies the attention mechanism multiple times, with separately learned weights, in parallel. For example, a multi-head attention block with 4

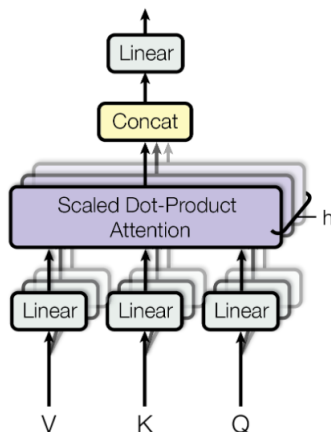


Figure 2.2: Multi-head attention. Adapted with permission from [37]

heads would learn 4 separate instances each of the W_q , W_k , and W_v weights. In parallel, 4 unique groups of Q , K , and V matrices are created and subjected to the dot-product attention mechanism described by equation 2.1. Then, the outputs are concatenated to be passed to the next stage of the model. This allows a finer resolution to be maintained while also permitting parallel computation and long-distance attention [37]. The multi-head attention mechanism is depicted visually in figure 2.2.

In order to easily compare models with similar numbers of parameters and similar compute requirements, the dimensionality of models employing multi-head attention is usually stated to be $d_{model} = n_{heads} \times d_{k/v}$ where $d_{k/v} = d_k = d_v$ as is common in most transformer models. In this way, models with identical d_{model} have similar compute requirements regardless of number of heads.

A transformer model is typically made up of many layers of attention mechanisms interspersed with feed forward layers (also known as dense or fully connected). The arrangement of such blocks to create a single “transformer layer” is dependent on the type of transformer in which it is found. Transformers can be encoder transformers [10, 43], decoder transformers [4], or both [37]. The main difference between encoder and decoder transformers is that decoders use auto-regressive modeling to generate output [26]. This means that for a language modeling task requiring text generation, the decoder (or encoder-decoder) will generate one piece of text at a time based on the input and on the pieces of text which have already been generated but not those which will be generated next [26, 37]. An encoder-only model does not do this and instead generates its output all at once [10]

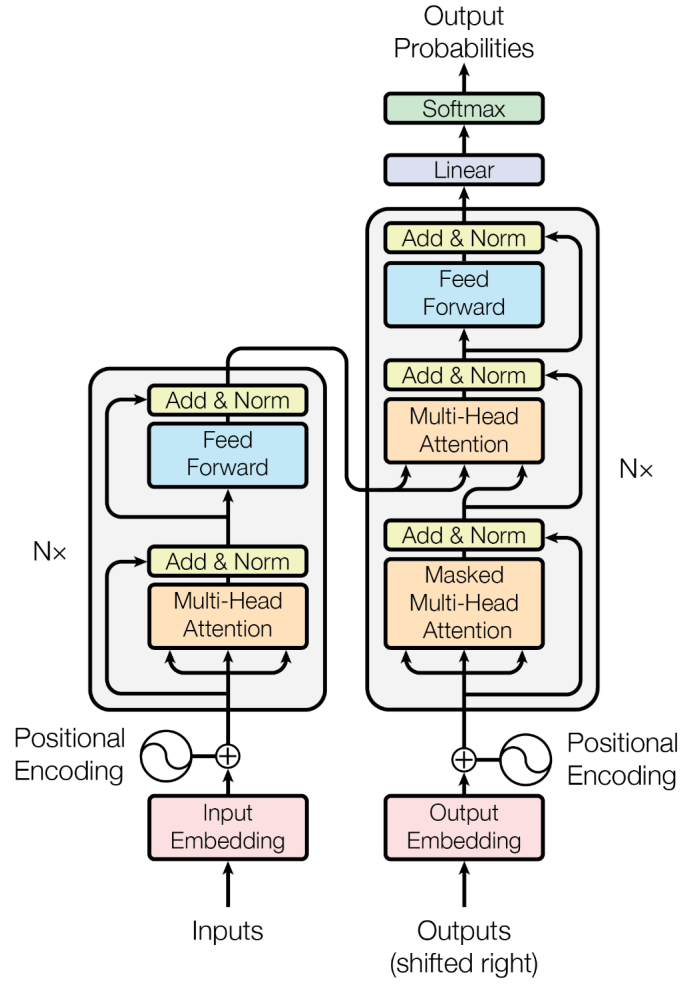


Figure 2.3: Transformer architecture proposed by Vaswani *et al.* The structure for an encoder block is displayed on the left while the structure for a decoder block is displayed on the right. Adapted with permission from [37]

which makes it more suitable for language tasks such as text classification rather than for text generation.

As shown in Figure 2.3, the original transformer designed by Vaswani *et al.* [37] included encoder blocks which consist of a multi-head attention layer followed by a single fully connected layer. Each of these are also bypassed by residual connections and followed by a normalization layer. The design of the decoder blocks for this original transformer is also displayed in Figure 2.3 but is not relevant to the rest of this thesis and will not be discussed further.

2.2 Language Transformers

Although transformers are used for sequence modeling tasks outside of language (including in this thesis) they were originally developed for natural language processing tasks [37]. As such, it is useful to briefly discuss some aspects of transformers in this light, before explaining how they are adapted for other tasks.

In order to convert text into a computable form, a tokenization strategy is employed. Tokenization breaks text into small sequences of letters called tokens [41]. For a given tokenization scheme, there will be a given number of tokens (for example 30,000) available to be assigned to sequences of characters. These sequences are chosen through optimization strategies to minimize the number of tokens required to express a given training set of natural language [41]. This leads to most tokens being either whole common words or sequences of characters which are often found together. For example, “they” and “ing” may be assigned a token each as those sequences should be common in natural language while “cacophony” or “xzrt” would likely be rare and unseen respectively and thus not be assigned tokens.

For each token, a vector embedding is then generated resulting in an embedding matrix representing all token embeddings [10, 37]. In order to embed a sequence of text, the text is split into the fewest tokens possible, then the embeddings for those tokens are used as input to the transformer model. However, other information is often encoded as well. For example, the encoder-only transformer model BERT [10] also uses positional and segmentation embeddings. The position embedding is unique for each token in an input sequence based on how many tokens come before it [10]. The segmentation embedding varies depending on which segment a token belongs to. For example, for text containing a question and an answer, the tokens which make up the question may be given a different segmentation embedding than those which make up the answer [10]. This is also particularly useful for

unsupervised pre-training. The three embeddings are summed for each token resulting in a final embedding [10]. The sequence of input embeddings is then passed as input to the first transformer layer of the model.

Since there exists such a large amount of freely-available unlabeled natural language data on the internet (such as from Wikipedia), unsupervised pre-training has been a large factor in recent advancements in natural language processing performance [4, 36, 9, 32, 35, 30, 26]. BERT [10] uses two distinct pre-training tasks. The first is masked language modeling in which some percentage of the input text is masked (hidden from the model). This is done within the scaled dot-product attention mechanism, as shown in Figure 2.1. The model must then predict what text was removed from the input data based on what text remains. The other main pre-training task is next-sentence prediction. For this task, BERT [10] is presented with two sentences and is asked to predict whether the second sentence actually follows the first as found in the dataset, or if the second sentence was randomly chosen from elsewhere in the dataset. The two distinct sentences are identified in the model’s input by using the same segmentation embedding value for the tokens which are part of the first sentence and a different segmentation embedding value for the tokens which are part of the second [10].

Tokenization, embedding, and unsupervised pre-training tasks are paramount to the successful application of transformers for natural language processing. In order to apply transformers to other domains such as biosignal processing, these practices must be specially adapted for them.

2.3 BIOT

BIOT [43] is an encoder-only transformer architecture which has adapted tokenization, embedding, and pre-training tasks for multichannel biosignals. The first step to tokenize such a signal is to divide each channel into overlapping windows of signal. Each of these windows is then treated as a token. For each token, a spectral embedding, a channel embedding, and a positional embedding is generated [43]. The channel embeddings are learned in an embedding table similar to segment embeddings in BERT [10]. However, in this case, there may be more than only 2 segments depending on the specific biosignal. For example, EEG data could have more than 20 channels [28]. The relative position embeddings are generated using the sin and cosine functions and are therefore not learned [43]. Finally, the spectral embedding is generated by first applying a fast-Fourier-transform (FFT) to the signal. Then, the resulting frequency energy vector is passed to a feed forward network to generate the final spectral embedding. This is BIOT’s [43] counterpart to BERT’s [10]

token embeddings. As is done for BERT [10], the 3 embeddings are summed together to create the final embeddings [43]. These are then passed as input to the transformer network for further processing [43]. The final step is to apply mean pooling across the encoded sequence such that the output of the model is a single vector with the same length as each input embedding was [43].

In order for BIOT [43] to leverage unsupervised pre-training as language transformers do, a pre-training task was developed for it. First, a biosignal sample is perturbed by dropping out a random subset of the sample’s channels and then also dropping out a random subset of the tokens in the remaining channels [43]. Then, both the perturbed signal and the intact signal are passed through BIOT to obtain a perturbed encoding and an intact encoding for the sample [43]. The task is then to recover the intact encoding by way of the perturbed encoding. A small feed forward network is thus used to transform the perturbed encoding into a predicted intact encoding [43]. Then, within a training batch, each intact encoding is compared through the dot-product operation with each predicted intact encoding. Cross-Entropy loss can then be applied to encourage dot-products which are between encodings from the same sample to be high and for dot-products which are between encodings from different samples to be low. This overall process has been used in other domains such as image processing and is known as contrastive learning [8]. The described objective function is shown below:

$$L = CrossEntropyLoss(Softmax(\langle Z, \tilde{Z} \rangle / T), I) \quad (2.2)$$

where L is loss, Z is the intact encoding, $\tilde{Z}$ is the predicted intact encoding, and T is temperature (set to 0.2) [43].

Yang *et al.* [43] show that this use of contrastive learning for pre-training their BIOT model is able to improve performance on downstream tasks. For example they show that BIOT can be pre-trained on datasets of resting EEG signals in order to improve its performance on seizure detection or classification tasks on separate EEG datasets [43]. Similarly, they show that pre-training on ECG signals improved performance on the task of arrhythmia detection in other datasets. Yang *et al.* [43] also showed that BIOT could outperform other models on HAR tasks using accelerometer data but did not choose to show that pre-training could further improve that performance.

For their tests with EEG data, Yang *et al.* [43] use their unsupervised pre-training method with a proprietary unlabeled resting EEG dataset of 5,110,992 10 second samples which they nickname PREST. They also experiment with jointly pre-training with another EEG dataset from the Sleep Heart Health Study (SHHS) [45] from which they extract 5,093,522 30 second samples of sleep EEG with only 2 channels each (of an available 16).

They then fine-tune these models on 4 different EEG datasets. These are one dataset gathered by an MIT PHD student at the Children’s Hospital Boston (CHB-MIT) [31] which has labels for binary classification between seizure or no seizure, the Interictal Ictal Injury Continuum Seizure dataset (IIIC Seizure) [14, 18] which has labels for seizure type classification between 6 types, Temple University Abnormal (TUAB) [34] which had labels for binary classification of abnormal or normal, and Temple University Events (TUEV) [2] which has labels for phenotype classification of EEG between 6 types.

For tests with ECG data, Yang *et al.* [43] use their unsupervised pre-training method with the ECG dataset from the 2020 Physionet Cardiology Challenge (Cardiology) [6]. After excluding the Physikalisch Technische Bundesanstalt-XL (PTB-XL) dataset [39], one of the 6 smaller datasets which make up Cardiology, Yang *et al.* [43] are left with 495,970 10 second ECG samples. They experiment with keeping 6 leads (channels) or 12 leads in their unsupervised pre-training. They then fine-tune these models on the PTB-XL dataset [39] on the binary prediction task of arrhythmia detection.

Finally, for the accelerometer tests, Yang *et al.* [43] train a randomly initialized BIOT model on the HAR dataset from [7] which includes labels for 6 different activity types, making this fine-tuning task multi-class classification.

The performance of each pre-trained model and randomly initialized BIOT [43] model is compared to that of other state-of-the-art randomly initialized models: SPaRCNet [18], ContraWR [42], CNN-Transformer [29], CNN-LSTM feature fusion network (FFCL) [24], and Spatial-Temporal Tiny Transformer (S3T) [33]. An abridged collection of results can be seen in Tables 2.1 and 2.2. Binary classification performances are reported with the area under the receiver operating characteristic curve (AUROC) metric. Multiclass classification performances are reported with Kohen’s Kappa. For both metrics, a higher score is better. A full summary of results can be found in [43].

Yang *et al.* [43] conclude by calling other researchers to pursue the development of large foundational biosignal models based on BIOT.

2.4 Maximal Update Parametrization

A challenge of building large foundational models in any domain is that of hyperparameter tuning and hyperparameter transfer. For example, learning rates may need to be reduced when increasing model width [44]. If this is not done, training can diverge, resulting in a very poorly performing model [44].

Table 2.1: Results from tests with BIOT from [43] trained on EEG data. Results on the TUAB dataset [34] are reported with AUROC scores while those for TUEV [2] are reported with Cohen’s Kappa scores.

Model	TUAB	TUEV
SPaRCNet [18]	0.8676 ± 0.0012	0.4233 ± 0.0181
ContraWR [42]	0.8456 ± 0.0074	0.3912 ± 0.0237
CNN-Transformer [29]	0.8461 ± 0.0013	0.3815 ± 0.0134
FFCL [24]	0.8569 ± 0.0051	0.3732 ± 0.0188
S3T [33]	0.8707 ± 0.0019	0.3765 ± 0.0306
BIOT [43] (Randomly Initialized)	0.8691 ± 0.0033	0.4482 ± 0.0285
BIOT [43] (PREST [43])	0.8730 ± 0.0021	0.4932 ± 0.0301
BIOT [43] (PREST [43] & SHHS [45])	0.8739 ± 0.0019	0.4841 ± 0.0309

Table 2.2: Results from tests with BIOT from [43] trained on ECG and accelerometer data. Scores for PTB-XL [39] are reported with AUROC scores while those for HAR [7] are reported with Cohen’s Kappa scores.

Model	PTB-XL	HAR
SPaRCNet [18]	0.7550 ± 0.0073	0.9236 ± 0.0189
ContraWR [42]	0.4433 ± 0.1557	0.8879 ± 0.0201
CNN-Transformer [29]	0.4996 ± 0.0936	0.8273 ± 0.0953
FFCL [24]	0.5127 ± 0.0051	0.8216 ± 0.0177
S3T [33]	0.6003 ± 0.0179	0.9213 ± 0.0076
BIOT [43] (Randomly Initialized)	0.74993 ± 0.0167	0.9351 ± 0.0160
BIOT [43] (Cardiology [6] 6)	0.7671 ± 0.0116	N/A
BIOT [43] (Cardiology [6] 12)	0.7659 ± 0.0076	N/A

In order to address this, Yang *et al.* [44] developed maximal update parametrization. They postulate that based on the central limit theorem, the sum of n iid samples with zero mean needs to be scaled by $1/\sqrt{n}$ for the sum to approach a non-zero and finite constant as n approaches infinity [44]. Given the many examples of dot-products in neural networks (such as in dot-product attention) they use this theorem to develop a strategy for scaling neural networks such that the results of all dot-products tend toward non-zero finite constants. This then allows the previously mentioned hyperparameters to maintain a constant optimal value across networks of varying sizes [44].

The results of Yang *et al.*'s [44] derivations show that both the learning rate and the initialization variance for each layer must be scaled individually according to that layer's shape. This can be implemented with a base learning rate and, if desired, a base initialization variance, in concert with per-layer multipliers for each [44]. The values of these multipliers are displayed in Table 2.3 and compared to standard parametrization. The base values of learning rate and initialization variance can then be kept constant while varying model size. In addition to per-layer learning rate and initialization variance multipliers, Yang *et al.* [44] find that dot-product attention in transformers must be scaled by $1/d$ rather than by $1/\sqrt{d}$ as is typically done and is shown in equation 2.1. Lastly, Yang *et al.* [44] assert that when training using the ADAM optimizer [22], if weight decay is being used then ADAMW [25] should be used instead of ADAM to scale weight decay correctly and maintain a proper implementation of maximal update parametrization [44].

This parametrization allows model developers to perform hyperparameter tuning on smaller models and then to directly transfer those hyperparameters to a larger model [44]. This process uses much less compute than hyperparameter tuning on the larger model itself and results in much better performance than guessing hyperparameters for larger models. This is imperative for extremely large models like the large language models of modern times as the compute budget for developing such a model typically only allows it to be trained once. A divergent training run would be a very costly waste of compute resources.

2.5 Training Compute Optimal Large Language Models

Recently, there has been a rush of research around increasing the size of language transformer models to improve performance [4, 9, 35, 36, 32, 26]. Along with these increases in model size, there have been no signs that researchers are approaching a performance limit for language [20]. The limitations preventing further improvement are thus only compute

Table 2.3: Maximal update parametrization (μP) and standard parametrization (SP) multipliers. μP varies depending on a layer’s input and output sizes (shown as *input* $\rightarrow$ *output* in the table). Here, *scale* represents a dimension of the model that is to be scaled up (which in a transformer would be d_{model} as defined in Section 2.1) while *const* describes a dimension that is not changed when the model is scaled up such as the input or output size in many cases. SP does not change for layers with different input or output shapes. All values are taken from [44].

parametrization	μP			SP
shape	<i>const</i> $\rightarrow$ <i>scale</i>	<i>scale</i> $\rightarrow$ <i>const</i>	<i>scale</i> $\rightarrow$ <i>scale</i>	all
initialization variance	$1/d_{in}$	$1/d_{in}^2$	$1/d_{in}$	$1/d_{in}$
SGD LR	d_{out}	$1/d_{in}$	1	1
Adam LR	1	$1/d_{in}$	$1/d_{in}$	1

budget and dataset size. Therefore, using a given compute budget as optimally as possible is imperative for best performance.

Hoffmann *et al.* [16] in ”Training Compute Optimal Large Language Models” run a battery of tests with language transformer models in order to study the best balance of model size versus number of training tokens for any compute budget. They assume power law relationships between compute, optimal number of parameters, and optimal number of training steps as defined by:

$$N_{opt} = a_n C^{b_n} \text{ [16]} \quad (2.3)$$

$$D_{opt} = a_d C^{b_d} \text{ [16]} \quad (2.4)$$

where N_{opt} is the optimal number of training tokens, D_{opt} is the optimal number of model parameters, C is the compute budget, while a_n , b_n , a_d , and b_d are constants. This work adopts these same equations to fit scaling laws in Chapter 4.

Hoffmann *et al.* [16] use 3 different methods to predict values for these constants. The first method involves training many models of different sizes for varying numbers of training tokens each.¹ Then, for a given compute budget, all of the saved training curves can be sampled at that compute budget in order to find the minimum pre-training loss [16]. The origin of that minimum loss would then indicate the optimal number of parameters and

¹Note that Hoffman *et al.* use a learning rate schedule. This means that the final loss of a model trained for 10 times fewer tokens than another model of the same size is not necessarily comparable to the pre-training loss of that model after the first 10% of its training has been completed. Thus, they need to train many models of the same size in order to model the effects of large changes in numbers of training tokens for the same model size.

number of training tokens for that compute budget [16]. By doing this for a range of 1500 logarithmically spaced compute budgets, they collect sufficient triplets of values for compute budget, optimal number of parameters, and optimal number of training tokens such that they could fit 2.3 and 2.4 to their results in order to generate a scaling law [16].

The second method they employ is to make use of what they call iso-FLOP curves [16]. They generate an iso-FLOP curve by training a set of models with different numbers of parameters with the same compute budget [16]. This means that larger models will be trained with fewer numbers of tokens than the smaller models such that a constant compute budget is used for each model. They then plot the final pre-training loss for each model against the number of parameters for each model. The resulting group of points are called an iso-FLOP curve. Once enough points are present such that there is a clear optimum where the final pre-training loss is at its minimum and thus the number of parameters and number of training tokens are at their most optimal values, this process is repeated for other compute budgets to generate more iso-FLOP curves [16]. The result is a set of optimal value pairs of D and N for a variety of compute budgets C . Equations 2.3 and 2.4 are then fit to these points to create a scaling law [16].

The third method they use is to fit a parametric loss function to the training curves of many models of different sizes trained for varying numbers of training tokens [16]. The parametric loss function they use is defined by:

$$L = \frac{N_C}{N^\alpha} + \frac{D_C}{D^\beta} + E \quad [16] \quad (2.5)$$

where E , α , β , N_C , and D_C are constants, L is loss, N is the number of training tokens, and D is the number of model parameters. The E term represents the minimum loss [16]. This is somewhat of an estimate of the random noise in the dataset which can never be predicted. Given the fitted parametric loss function they are able to create scaling laws in the forms of equations 2.3 and 2.4 by minimizing the loss function for arbitrary compute budgets under the constraint $F = 6ND$ where F is the number of FLOPs for a compute budget [16]. This is an estimate of number of FLOPs used for training defined by Kaplan *et al.* [20].

The first method they use results in a scaling law with $D \propto C^{0.5}$ and $N \propto C^{0.5}$, the second method $D \propto C^{0.49}$ and $N \propto C^{0.51}$, and the third $D \propto C^{0.46}$ and $N \propto C^{0.54}$ [16]. The scaling law which resulted from the first method was tested against Gopher [30], a large language model with a compute budget for training that was approximately 100 times larger than the largest budgets used to create the scaling law. By extrapolating the optimal number of parameters and training steps for a compute budget that large, Hoffmann *et al.* [16] predicted that training a model that was 4 times smaller for 4 times

as many training tokens would result in a better performance. Such a model, given the name of Chinchilla when trained by Hoffmann *et al.* [16], did indeed outperform Gopher. This indicated that the scaling law was more accurate than whatever heuristic was used to determine the allocation of compute budget for Gopher. Although, this does show that the scaling law was useful, it does not completely show how accurate it was as no test to verify that a model which was even smaller than Chinchilla and trained for more tokens would perform worse or better [16]. The reason Hoffman *et al.* [16] did not complete such a test or report results from multiple different seeds was due to compute budget restraints. However, they claim that the trends they observe are clear and that noise seems to be minimal [16].

2.6 Scaling Laws for Neural Language Models

In addition to scaling laws for optimal compute budget allocation, there have been many other aspects of large language models for which scaling laws have been developed. For example, in “Scaling Laws for Neural Language Models”, Kaplan *et al.* [20] run many experiments focused on a diverse array of model parameters and their relationships with test loss for language modeling transformers. They test primarily with decoder-only transformers [20]. One of the main conclusions they draw from their tests is that performance is much more correlated with scale than with other factors. Specifically, the dataset size, number of learnable non-embedding model parameters, and training compute budget have strong influences on performance while the shape of a model (depth, width, head size) has much less effect if it is kept within a reasonable range [20]. For instance, to avoid a loss increase of greater than 2% over the optimal ratios, d_{model}/n_{layer} should be kept between 7 and 200 while d_{model}/n_{heads} should remain under 256^2 [20]. This finding can improve future language modeling performance by encouraging model developers to re-allocate some of their compute budget away from hyperparameter tuning for model shape and towards increasing the scale of their tuned model.

The experiments surrounding dataset size, number of learnable parameters, and training compute budget show that each has a power law relationship with test loss when they are the limiting factor [20]. For example, if the number of training steps and number of parameters are sufficiently large, but the dataset size is small (hence limiting performance by causing overfitting) then the resulting test loss is predominantly controlled by the dataset

²The smallest head size they test is 16 which does not increase loss by more than 2%

size and that relationship can be modeled by a power law given by:

$$L(N) = \left(\frac{N_C}{N}\right)^{\alpha_N} \quad [20] \quad (2.6)$$

where L is loss, N is number of tokens in the dataset, and N_C and α_N are constants. Kaplan *et al.* [20] further study the effects of dataset size by relating performance to a function of both dataset size and model size. They first assume a relation in the form of:

$$L(N, D) = \left[\left(\frac{D_C}{D}\right)^{\frac{\alpha_D}{\alpha_N}} + \frac{N_C}{N} \right]^{\alpha_N} \quad [20] \quad (2.7)$$

where D is the number of parameters in the model, D_C and α_D are constants, and L , N , N_C , and α_N are the same as in equation 2.6. Once the constants were fit to experimental data, Kaplan *et al.* [20] were able to determine that, for the models and datasets they were experimenting with, for every doubling of model size, the dataset size need only be increased by approximately 74% to avoid any new overfitting.

Another one of Kaplan *et al.*'s [20] important findings is that test performance is a good predictor of performance on unseen datasets. This indicates that it can be predicted with a high degree of accuracy that models which are shown to perform best on a given dataset will also perform best on another dataset when applied to it via transfer learning. This finding can be applied to streamline model training for transfer learning by eliminating any need to test multiple pre-trained models on the target dataset.

Additionally, Kaplan *et al.* [20] find that not only can power laws be used for predicting final loss based on scale factors, but can also be used to extrapolate training curves to predict train loss for models at future parameter update steps. The power law which they fit to their training curves is:

$$L(D, S) = \left(\frac{D_c}{D}\right)^{\alpha_D} + \left(\frac{S_c}{S_{min}(S)}\right)^{\alpha_S} \quad [20] \quad (2.8)$$

where L is loss, D is the number of non-embedding parameters in the model, and S_{min} is the minimum number of steps required to reach a given loss value. This assumes a very large batch size. D_c , α_D , S_c , and α_S are constants.

Finally, Kaplan *et al.* [20] also extrapolate from their empirical results their own scaling law to predict the optimal balance of model size and number of processed tokens. Contrary to Hoffmann *et al.* [16] it was found that the number of processed training tokens should only double for every five-fold increase in model size rather than growing in equal proportion

to model size. Hoffmann *et al.* [16] attribute this to their use of a learning rate schedule which they show leads to better final loss. This discrepancy indicates that optimal compute budget predictions have the potential to vary drastically for different training strategies. Thus, in order to increase the scale of the BIOT model, which is different than language models in some ways as discussed in Section 2.3, a scaling law should be made specifically for it.

Chapter 3

Experiment Setup

This chapter will describe work which was done to prepare the experiments used to generate scaling laws in the next chapter. Topics covered include documentation of the datasets used, the implementation of maximal update parametrization for [BIOT](#), hyperparameter selection, and an exploration of possible overfitting with different levels of data repetition.

3.1 Datasets

3.1.1 National Health and Nutrition Examination Survey

[National Health and Nutrition Examination Survey \(NHANES\)](#) [1] is an ongoing set of surveys and studies undertaken by the Centers for Disease Control and Prevention (CDC) in the United States aimed at tracking indicators of health among the American populace as well as recording data that could be useful to researchers in developing health recommendations. From 2011 to 2014, the physical activity measurement (PAM) component of [NHANES](#) was recorded using an accelerometer worn on the non-dominant arm of participants [1]. The participants were asked to wear the accelerometer for 8 consecutive days before returning it to the study organizers [1]. For the [NHANES](#) cycle of 2013-2014 there is PAM data available for 7,776 participants [1].

The accelerometer data was recorded continuously at 80Hz for x, y, and z axes while the participants lived their daily lives [1]. For each participant there are up to 194 hours of data available but in some cases less depending on when the participant removed their monitor. The [NHANES](#) researchers used the open source Monitor-Independent Movement

Summary (MIMS) [19] project to label each minute of data as one of “active-wear”, “sleep” or “non-wear” [1]. They also include a certainty score generated by MIMS with each minute [1].

In order to avoid training on extended periods of sleep or non-wear, I used the available labels to filter the data. Specifically, since the data is stored in 1-hour files, I deleted all 1-hour files which did not have all 60 minutes labeled as “active-wear”. This reduced the number of hours in the dataset from just under 1,500,000 to 296,846, retaining almost 20% of the dataset. Although the MIMS confidence score was available, it was not used since some small contamination of non-wear or sleep time was unlikely to impact training.

The dataset was then split into 285,922 hours of training data and 10,924 hours of validation data. The training and validation sets were made up of accelerometry from different subjects only. Note that some tests were done with a smaller amount of training data before the full dataset was downloaded. This earlier version had only 95,756 hours but was still only composed of 100% “active-wear” samples. Results from training on this smaller dataset are only shown in Figures 3.3 and 3.4. Some training runs are also done with this smaller dataset in Section 4.1.1 as documented there in Table 4.1.

3.1.2 Temple University Hospital EEG

The Temple University Hospital EEG (TUH-EEG) [28] corpus is a massive dataset of 69,652 (at the time of writing) EEG recordings from thousands of patients. The samples are heterogeneous in recording standards. Specifically, different channels, reference voltages, and sampling rates are used throughout the corpus. Upon analyzing the dataset, I found that a subset of 24 channels with a common reference were found in 49,528 of the available EEG recordings. These 24 channel names, as they appear in the dataset, are shown in Appendix A. Furthermore, the most common sampling rate in the dataset was 256Hz. For this reason, all EEG recordings which were not at this sampling rate were resampled. Note that the lowest sampling rate was 250Hz with some as high as 1000Hz.

Due to storage constraints, of the 49,528 EEG recordings with the chosen channels, only 30,063 were kept for a total of approximately 12,073 hours of EEG data. This is split into 29,063 train recordings and 1000 validation recordings. The validation set has nearly¹ no samples from patients in the training set.

¹The validation set was created by removing the last 1000 recordings from the dataset. The dataset is organized such that recordings from the same patient are grouped. One patient appears in the validation set and the training set due to this.

3.1.3 Data loading

Both datasets were split into non-overlapping samples of 10 seconds long each for the hyperparameter tuning done in Section 3.3.4, the overfitting tests done in Section 3.4, Figure 3.5 and in the development and testing of the scaling laws which were found to be accurate in Sections 4.2, 4.3, 4.4, 4.5.

However, for the first attempt at developing a scaling law for the NHANES dataset using the iso-FLOP approach in Section 4.1, and for the tests shown in Figures 3.3 and 3.4 a bug was present in the code responsible for loading the datasets. This resulted in successive samples being taken with a shift of only one time step rather than by 10 seconds. Thus, for each file (equivalent to 1 hour in NHANES [1]), a number of samples equal to the true potential number of samples in the file would be taken only from the beginning rather than from the entire file. For example, the NHANES dataset [1] is organized into 1 hour long recordings and is resampled to 80Hz. Thus, instead of 360 non-overlapping samples, that file would be split into 360 samples between the 0 second and 14.5 second marks each shifted by 1/80 of a second.

This bug was discovered after all experiments in this thesis had already been run. Upon re-running the hyperparameter tuning done in Section 3.3.4, the overfitting tests done in Section 3.4, and the development and testing of the scaling laws which use the full datasets and were found to be accurate in Sections 4.2, 4.3, 4.4, 4.5, it was found that results were widely similar and qualitatively the same as previously save for some quantitative changes in the resulting scaling laws. In order to avoid further exceeding this project’s budget and to allow the reader to enjoy the insights into maximal update parametrization [44] provided by Figures 3.3 and 3.4 as well as insights into the process of effectively implementing the iso-FLOP method [16] provided by Section 4.1, these results are left in this document along with this documentation of the sub-optimal data loading method which was applied.

3.2 Maximal Update Parametrization for BIOT

Maximal update parametrization [44] was implemented for BIOT [43] to allow for constant hyperparameters to be used across different model sizes as was done by Hoffmann *et al.* in [16]. An important benefit of maximal update parametrization is that larger models always perform better (or equal to) smaller models, which is a result observed empirically and explained theoretically by Yang *et al.* [44] If larger models were not guaranteed to have better performance, a scaling law to predict the best balance of model size and

training steps for larger compute budgets would not be useful. This section describes the implementation and verification of maximal update parametrization for BIOT.

3.2.1 Implementation

Maximal update parametrization was implemented largely with the use of the mup Python package [17] which was developed by Yang *et al.* [44] when they derived the requirements for maximal update parametrization. The steps taken to modify BIOT were as follows:

- **Appropriate weight initializations were modified.** As described in Table 2.3, standard parametrization uses $1/d_{in}$ for initialization variance for all layers. The only difference for maximal update parametrization is that layers which have an input dimension which scales with width and an output dimension which does not, should use $1/d_{in}^2$ as their initialization variance [44]. Since BIOT is an encoder only transformer [43], it does not have any layers that map from a scaled dimension back to a constant dimension. This is because such layers are typically output layers which must provide an output in some fixed shape regardless of model size. Since BIOT only outputs an encoding during pre-training, its output layers map to a scaled dimension and do not need to be initialized differently than in standard parametrization.

Thus, the only change that was made to the implementation of BIOT’s weight initialization was to add a common tunable multiplier to the already-used initialization variances as recommended by Yang *et al.* [44].

- **The standard Adam [22] optimizer was replaced.** The original BIOT [43] architecture uses the base Adam optimizer for training. There is an implementation of Adam in the mup Python package [17] which automatically scales learning rates based on the shapes of layers. However, BIOT [43] also uses weight decay. Yang *et al.* state that AdamW [25] is necessary for correct scaling of weight decay, which I confirm in Section 3.2.3. For that reason, the base Adam optimizer was replaced with the mup implementation of AdamW [17] which scales both learning rate and weight decay correctly. Note that the objective function which was used for optimization was Equation 2.2 as was used by Yang *et al.* [43]
- **Base shapes were set for maximal update parametrization.** In order for the mup package to scale hyperparameters such as learning rate and weight decay correctly, a base model shape must be set [17]. I set a BIOT model with $d_{model} = 256$ and $n_{layers} = 4$ as the base model. The hyperparameters which are scaled by

maximal update parametrization are kept at their base values for this shape and scaled based on the difference in shape between this base model and other models that hyperparameters are being transferred to.

- **Attention scaling was modified.** As prescribed by Yang *et al.* [44] attention scaling was changed from $\frac{1}{\sqrt{d_{model}}}$ to $\frac{1}{d_{model}}$. See equation 2.1 for the original attention scaling.

3.2.2 Coordinate checks

The mup package provides a method for verifying a given maximal update parametrization implementation for correctness. The method is referred to as coordinate checking [17]. The process involves monitoring the (pre-)activation outputs of each layer within a neural network during a few steps of training to ensure they do not explode as training progresses [17]. This is then done for many model sizes.

A successful coordinate check will see the l_1 norm of the (pre-)activation outputs of each layer remain constant across successive training steps and different model sizes [17]. An example can be seen in Figure 3.1. An unsuccessful check will observe some of these norms deviate from constant as model size increases—indicating that maximal update parametrization is not implemented correctly [17]. An example can be seen in Figure 3.2.

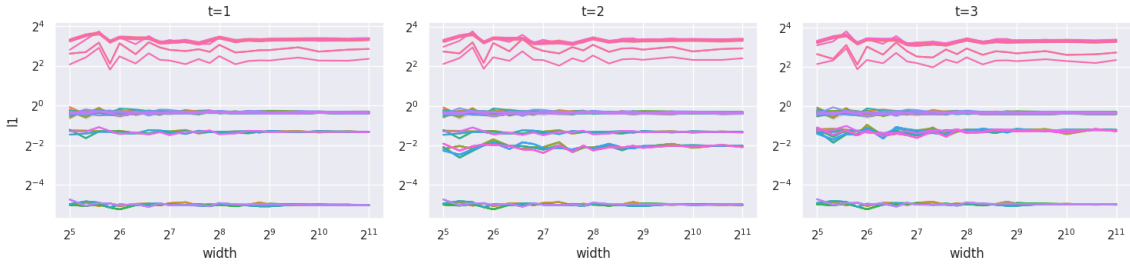


Figure 3.1: Coordinate check for a correct maximal update parametrization implementation for BIOT. The y axis titled l_1 represents the l_1 norm of the (pre-)activation vectors. The x axis titled width represents the value of d_{model} . Each plot ($t=1, 2, 3$) represents the state of (pre-)activation vectors after 1, 2, and 3 steps of backpropagation from left to right as displayed. Although a legend is not supplied due to the number of curves, each represents a single (pre-)activation vector.

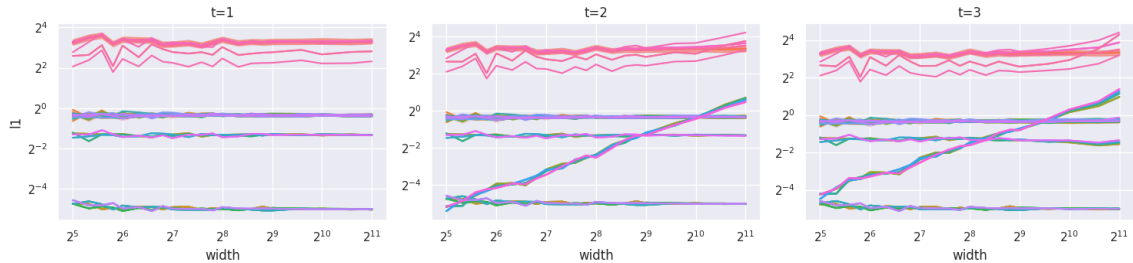


Figure 3.2: Coordinate check for an incorrect maximal update parametrization implementation for BIOT. In this case, the base Adam was used instead of the mup implementation of AdamW. But other changes listed in Section 3.2 are implemented. The plot and axes titles are identical to those in Figure 3.1

The coordinate check displayed in Figure 3.1 was generated from the final implementation of maximal update parametrization for BIOT using the accelerometer configuration and data from NHANES. The EEG configuration is identical in terms of maximal update parametrization implementation and thus a separate coordinate check is not shown.

3.2.3 Upscaling verification

Another indicator of an incorrect implementation of maximal update parametrization is when increasing model size decreases performance for the same number of steps [44, 17]. As mentioned in Section 2.4, maximal update parametrization guarantees that this will not be the case when transferring hyperparameters. Thus, to further confirm that my implementation of maximal update parametrization was correct, I trained one model each with d_{model} set to 64, 128, 256, and 512 all with identical hyperparameters and training datasets. An example of an upscaling verification test which shows an incorrect maximal update parametrization is shown in Figure 3.3 while the verification from my final correct implementation of maximal update parametrization can be seen in Figure 3.4. Note that these tests use the bugged data loader described in Section 3.1.3. Training runs done with the correct data loader also show a correct maximal update implementation similar to Figure 3.4. But, no tests were done with both an incorrect maximal update implementation and a correct data loader implementation, so Figures 3.3 and 3.4 are both left for easy comparison between them.

Based on these upscaling verification tests, it is clear that maximal update parametriza-

tion is very useful for scaling artificial neural network models as well as developing scaling laws for such networks. It also demonstrates that my implementation of maximal update parametrization [44] for BIOT [43] is correct as both the upscaling verification test and the coordinate checking test were completed successfully.

3.2.4 Deficiencies at small sizes

Yang *et al.* [44] state that hyperparameter transfer works best past some minimum d_{model} . For their tests, which are on language transformers, they find that this minimum is approximately $d_{model} = 256$ [44]. Although I do not perform sufficient tests to discover with certainty where this point is for BIOT [43], it is reasonable to be wary of this reported issue as I am training models in and about that range.

Although it was shown that maximal update parametrization [44] was successful in ensuring that larger models were consistently improved through further size increases, it was observed that at small sizes this was not always the case. An example of this behavior can be seen in Figure 3.5. In this case, a model with $d_{model} = 32$ and a model with $d_{model} = 40$ perform approximately equally with the smaller model outperforming the larger for a few ranges of steps. Both models are outperformed by $d_{model} = 48$ and in turn $d_{model} = 56$. Models with $d_{model} = 64$ and $d_{model} = 72$ improve further but perform nearly identically. Finally $d_{model} = 80$ and $d_{model} = 88$ perform best but nearly equally to each other.

By again observing Figure 3.1 where the coordinate check [17] done for the maximal update parametrization [44] implementation used for these tests was shown, it is clear that the l_1 -norm of the (pre-)activation vectors of the smallest models are more noisy than those for the larger models. This supports the claim that scaling at smaller sizes could result in hyperparameter transfer that is suboptimal.

Note that this discrepancy is also observed in larger models (for example, in the range of $d_{model} = 400$) but only for much smaller relative size changes. This is attributed more so to general noise than to a deficiency of maximal update parametrization whereas the more significant discrepancy at smaller model sizes is attributed to a deficiency of maximal update parametrization in addition to general noise.

3.3 Hyperparameter Selection

In order to develop a scaling law which predicts the optimal balance of model size and number of training tokens, hyperparameters needed to be chosen such that they were equally optimal for each model size used in experiments. Note that consistency across model sizes was treated as more important than strict optimality. This section documents the choices made for each relevant hyperparameter to the scaling law experiments.

3.3.1 Shape

There are 3 relevant hyperparameters which are related to shape for BIOT [43]. These are n_{layers} , n_{heads} , and d_{model} . n_{layers} was set to a constant 4 for all models. n_{heads} was set such that d_{model}/n_{heads} (the size of each head) remained at a constant 8. d_{model} was the dimension which was varied in order to upscale the number of parameters in the model. It is varied from a minimum of 32 to a maximum of 512 for the training runs used to generate scaling laws. This dimension was chosen to be varied rather than all 3 because Yang *et al.* [44] found that other hyperparameters transfer more reliably with maximal update parametrization across different values of d_{model} than values of n_{layer} or d_{model}/n_{heads} .

n_{layers} was chosen to be 4 since that was the number of layers originally used for BIOT by Yang *et al.* [44] with a value for d_{model} of 256. Note that Levine *et al.* [23] find that for language transformers with more than 1 million parameters, 6 layers is optimal while up to 12 layers are optimal passed approximately 3 million parameters. For reference, a BIOT [43] model with $d_{model} = 144$ and $n_{layers} = 4$ has approximately 1 million parameters while changing d_{model} to 248 increases the parameter count to approximately 3 million. However, the differences in loss that were found by Levine *et al.* [23] were very minimal as was also the case in the work of Kaplan *et al.* [20] Furthermore, Yang *et al.* [44] state that their maximal update parametrization works best across width. Therefore, it was decided that maintaining a consistent depth was more advantageous than not in terms of consistency for developing a scaling law.

d_{model}/n_{heads} was chosen to be 8 such that it would be possible to vary d_{model} by increments of 8 while keeping the head size constant. As mentioned in Section 2.6, Kaplan *et al.* [20] found that the head size should remain under 256 which is satisfied by setting it to 8. Although, the lowest head size tested by Kaplan *et al.* [20] was 16, no signs of hindrance were observed when comparing training with a head size of 32 rather than 8 when experimenting with BIOT [43]. Overall, the benefit of being able to vary d_{model} by increments as

small as 8 for scaling law development was sufficient compensation for any potential small performance degradation brought on by a sub-optimal choice of head dimension.

Section 4.1.1 is an exception to this as head size was initially set to 32. Results in that section made the need to vary d_{model} by steps of 8 clear which is why all other results are obtained from models with head sizes of 8.

3.3.2 Preprocessing related

Three hyperparameters related to preprocessing were potential candidates for hyperparameter tuning. These were the sampling frequency for each dataset, the window length for tokenization, and the overlap of windows for tokenization. In early tests, these hyperparameters were included in hyperparameter tuning runs with Optuna [5]. However, optimal values varied significantly between studies. For example, the window lengths used for different configurations in Sections 4.1.1 and 4.2.1 varied from 72 to 152. Thus it was determined that the values of these hyperparameters were less critical than those of others. As such, they were excluded from further Optuna [5] studies in order to allow for more effective tuning of other hyperparameters. The three preprocessing hyperparameters were thus chosen to be simple, convenient, and close to those used by Yang *et al.* [43] for the original BIOT.

The chosen sampling frequencies were left at 80Hz for NHANES [1] and 256Hz for TUH-EEG [28] as described in Section 3.1. Yang *et al.* [43] used 200Hz for their EEG data and 50Hz for their accelerometer data which are similar enough that it was not deemed necessary to do further resampling. For window length, Yang *et al.* [43] choose to use 200 points for EEG data and 100 points for accelerometer data. This corresponds to windows of 1 second for EEG data and 2 seconds for accelerometer data. Since I selected different sampling rates, I chose 256 point windows for EEG and 80² point windows for accelerometer such that each had an effective window length of 1 second. Yang *et al.* [43] use an overlap of 50% for each EEG window and an overlap of 80% for each accelerometer window. I choose to use an overlap of 50% for each for simplicity.

²Sections 4.1.1 and 4.2.1 include models trained before the window size was set to 80 points in this way. As such some models use window sizes of 152 or 72. No significant difference in performance was observed. Overlap and sampling frequency were set to the values reported in this section for all tests.

3.3.3 Batch size

Since the loss used to train BIOT [43] is contrastive loss [8], the loss is dependent on the batch size. This is because the predicted encodings are compared to each intact encoding to ensure that for a given signal, the predicted encoding is similar to the intact encoding for the same signal but dissimilar to the intact encodings for all other signals within the batch. Thus, the loss is larger for larger batch sizes as there are more encodings for which each predicted encoding must be dissimilar. Therefore, it is difficult to compare different batch sizes to determine which is best based on pre-training loss alone.

Therefore, the batch sizes for both EEG and accelerometer data were chosen such that the largest models which were planned to be tested could be trained on available hardware without splitting batches. Yang *et al.* [43] used batch sizes of between 1024 and 2048. I initially used a batch size of 64 for accelerometer data, expecting to train models with d_{model} as large as 2048. However, after the first scaling law was developed in Section 4.1 it became clear that such a large model would not be required. Thus, the batch size for NHANES [1] was increased to 256 for further tests in Section 4.2. The batch size for EEG data was set to 100 for all tests. Note that the additional channels for EEG data significantly increased the compute requirement for each sample.

3.3.4 Optimizer related

The mup-AdamW [17] optimizer was used for all training runs. The hyperparameters related to it which were tuned were the base learning rate and weight decay. These were chosen for each dataset separately through Optuna [5] studies. Each study ran 10 trials with 1 epoch of training data for TUH-EEG and 100,000 steps (approximately 0.3 epochs) of training data for NHANES using models with $d_{model} = 256$. The objective was the minimization of the loss on the validation sets (as described in Section 3.1). 10 trials, rather than more, were used due to the long training time for a single model (2.5 hours for TUH-EEG and 5 hours for NHANES) to complete one trial. Furthermore, noise between successive models with new random initializations (can be observed for NHANES in Figure 4.4) made discovering perfectly optimal hyperparameters challenging. As was the case for other hyperparameters, it was decided that perfect optimality was not necessary for the development of a scaling law. Thus 10 trials were deemed to be sufficient to ensure both training convergence and effective hyperparameter transfer using maximal update parametrization.

The only hyperparameters tuned in the studies for each dataset were the base learning

rate and the weight decay concurrently.³ The bounds used for base learning rate were 10^{-5} to 10^{-2} and the bounds used for weight decay were 10^{-6} to 10^{-3} . For the [NHANES \[1\]](#) dataset with a head size of 8 and a batch size of 256, the base learning rate was 0.001132 and the weight decay was 0.00000371. For the [NHANES \[1\]](#) dataset with head size of 32 and batch size of 64, the base learning rate was 0.0001295 and the weight decay was 0.0002718. For the [TUH-EEG \[28\]](#) dataset the learning rate was 0.0000945 and the weight decay was 0.00000317.

3.4 Insights into Overfitting

Hoffman *et al.* [16] create scaling laws to predict the optimal balance of number of training tokens and number of parameters in language transformers by comparing only training loss rather than using validation or test loss. They do this since they are operating in the infinite data regime in which they do not repeat any training tokens during training. As such, training loss is an unbiased estimator of validation loss [16] (if the validation set is a sample of the same distribution as the training set) and thus overfitting cannot occur. Unfortunately, it was not possible to use datasets that were large enough to sufficiently train models with only 1 epoch of data for my experiments due to availability of data and limited storage capabilities.

However, using validation loss to create such a scaling law for [BIOT \[43\]](#) would be impractical. This is because the validation loss would need to be sampled at too many steps throughout training which would severely increase the computational cost of developing the scaling law.⁴ Furthermore, developing a scaling law where overfitting is a factor would not be useful. If overfitting is occurring, then more data should be collected before further compute budget increases. Therefore, I developed a test to provide some insight on the risks of overfitting in the training runs used to develop and test the scaling laws. If no overfitting is indicated, then it should be the case that the training loss curves in Chapter 4 are very similar to training curves which would be obtained with infinite data.

³With the exception of some studies for [NHANES \[1\]](#) training runs that included window size before that hyperparameter was set to 80 points manually.

⁴For the iso-[FLOP](#) approach [16] this could be feasible if the compute budgets for the curves were known in advance. But in practice, it is much more convenient to use training loss so that the iso-[FLOP](#) curve compute budgets can be chosen and modified based on the locations of the presumed optimal points. For the parametric loss function approach [16], validation loss would have to be sampled thousands of times to get the same number of samples as with using training loss.

Table 3.1: The results from the overfitting tests done with the [NHANES \[1\]](#) dataset. The number of epochs also indicates the factor used to shrink the dataset. For example, the results with 4 epochs are trained on one quarter of the full dataset for 4 epochs. Each entry in the same column is run with a different random seed but the same sample of the dataset. The value of each entry corresponds to the total sum of contrastive loss on the validation set.

number of epochs	1	4	16
$d_{model} = 512$	17500	17700	17600
	17600	17700	17700
	17600	17600	17700

3.4.1 Tests

For the [NHANES \[1\]](#) dataset, I trained a model with $d_{model} = 512$ with different fractions of the dataset but all for approximately the same number of steps. Specifically, I trained one model with the full dataset for 1 epoch. I then trained another model using only a quarter of the dataset repeating for four epochs. I did this again with one sixteenth of the dataset repeating for sixteen epochs. I then did all three of these tests for three random seeds each. Note that in the cases where the dataset was reduced, the reduction was done such that the number of samples from each subject was reduced equally rather than removing all data from a subset of subjects and keeping all data from others.

Each trained model was then evaluated on the [NHANES \[1\]](#) validation set. Theoretically, overfitting would be most likely to occur in the model trained on only one sixteenth of the dataset. If overfitting did occur, the validation loss for those models would be higher than for the others. Results of these tests could then be used to make predictions about the potential presence of overfitting in the training runs done in Chapter 4.

The same procedure was done for the [TUH-EEG \[28\]](#) dataset. However, in addition to tests with $d_{model} = 512$, another battery of tests with the same dataset reduction regime was done with models which had $d_{model} = 128$.

3.4.2 Results

The results from these experiments are shown in Table 3.1 for the [NHANES \[1\]](#) dataset and Table 3.2 for the [TUH-EEG \[28\]](#) dataset.

The validation loss for the [NHANES \[1\]](#) is lower by a mean of approximately 100 (or 0.5%) for the tests done with 1 epoch versus those done with 4 or 16 epochs. This indicates

Table 3.2: The results from the overfitting tests done with the [TUH-EEG](#) [28] dataset.

number of epochs	1	4	16
$d_{model} = 128$	1299	1261	1305
	1265	1239	1266
	1276	1243	1252
$d_{model} = 512$	1195	1182	1199
	1220	1204	1225
	1205	1193	1210

that there may be some minor overfitting for these levels of data repetition at a model size of $d_{model} = 512$. Although, a sufficient number of tests for statistical significance is not available due to compute budget restraints.

The validation loss for the [TUH-EEG](#) [28] dataset with $d_{model} = 128$ does not appear to be different between the runs with one epoch with the full dataset versus four epochs with one quarter of the dataset and sixteen epochs with one sixteenth of the dataset. This does not indicate overfitting at this model size and level of data repetition.

The validation loss for the [TUH-EEG](#) [28] dataset with $d_{model} = 512$ does show some potential overfitting for one sixteenth of the dataset repeated sixteen times. However, none is apparently indicated by the validation loss at one quarter of the dataset repeated 4 times. This indicates that some minor overfitting may occur for that level of data repetition at a model size of $d_{model} = 512$ but not for a data repetition level of four times.

3.4.3 Discussion

Sections 4.2 and 4.3 successfully develop scaling laws for the [NHANES](#) [1] dataset. None of the training runs used to develop the scaling laws use more than one epoch of the full dataset. There is therefore no chance of overfitting occurring in the development of the scaling law and no prediction about overfitting made from the results in this section are required. However, the models chosen to be smaller than the optimal models used when testing the scaling laws in Sections 4.2.3 and 4.3.3 are trained for up to 7 full epochs of the whole dataset. Thus, overfitting is possible in these cases. The values of d_{model} for these models are 144 and 152.

It was reported that Kaplan *et al.* [20] found that additional overfitting for a 100% increase in model size could be avoided by only a 74% increase in dataset size. Although these values cannot necessarily be applied to domains other than language processing, the

finding that larger models need larger dataset sizes to avoid overfitting should generalize to the training done in this thesis. In the tests done above, only minor overfitting was observed when using one quarter or one sixteenth of the [NHANES \[1\]](#) dataset with a model which had $d_{model} = 512$. Thus, the smaller models trained on the entire dataset repeated for 7 epochs are likely to experience less or even no measurable overfitting. Even overfitting at the level seen in these tests should be negligible for the tests done to validate the scaling laws.

In Sections [4.4](#) and [4.5](#), scaling laws are successfully developed using the [TUH-EEG \[28\]](#) dataset. The models with $d_{model} = 96$ and less are trained for 11 epochs with the full dataset while other models up to and including a size of $d_{model} = 128$ are trained for 3 epochs and other up to a size of $d_{model} = 512$ for 2 epochs. Since the tests reported in [Table 3.2](#) show no overfitting for a model with $d_{model} = 512$ on one quarter of the dataset repeated 4 times, it is unlikely that similar or smaller models experience significant overfitting when the full dataset is repeated for only two epochs. The tests reported in [Table 3.2](#) also showed no sign of overfitting for a model with $d_{model} = 128$ when one sixteenth of the dataset was repeated sixteen times. Thus, it is unlikely that similar or smaller models experienced overfitting when the full dataset was used to train them with 11 or 3 epochs.

However, the smaller models used for testing the developed [EEG](#) scaling laws at larger compute budgets are trained for approximately 40 epochs with the full dataset. These models have sizes of $d_{model} = 72$ and $d_{model} = 88$. It is unclear based on the tests done in this section whether overfitting is likely to occur in those models or not. It should thus be taken into consideration that the final loss of those models with no training data repetition may be underestimated by the final training loss as reported. The optimal test models were still able to outperform these smaller test models despite this possibility. In other words, if any overfitting was corrected for, the loss would be higher for the smaller test model which would make the correctness of the scaling law more compelling.

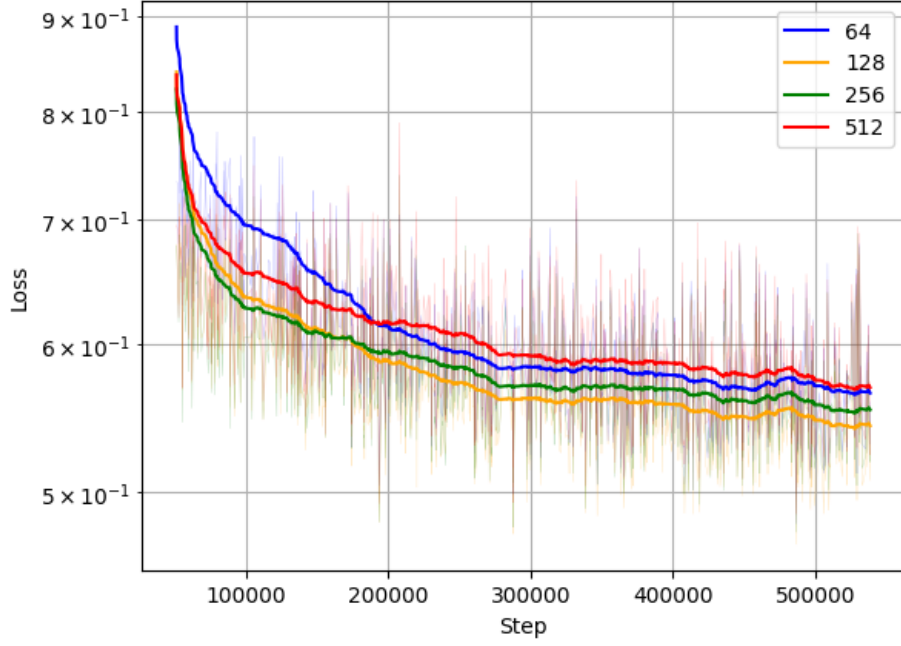


Figure 3.3: An example of an upscaling verification test which indicates an incorrect implementation of maximal update parametrization. This test was done for an implementation which was using mup-Adam [17] rather than mup-AdamW [17]. Note that this same maximal update implementation passed the coordinate check verification and was only discounted as incorrect during upscaling verification. The legend indicates the value of d_{model} for each training loss curve. This is the only difference between models for each curve. Note that these models were trained on the smaller NHANES [1] dataset described in Section 3.1.1, a batch size of 64, and with the incorrect data loader implementation described in Section 3.1.3. Their final loss values are thus not directly comparable to other runs with the full dataset or with a different batch size.

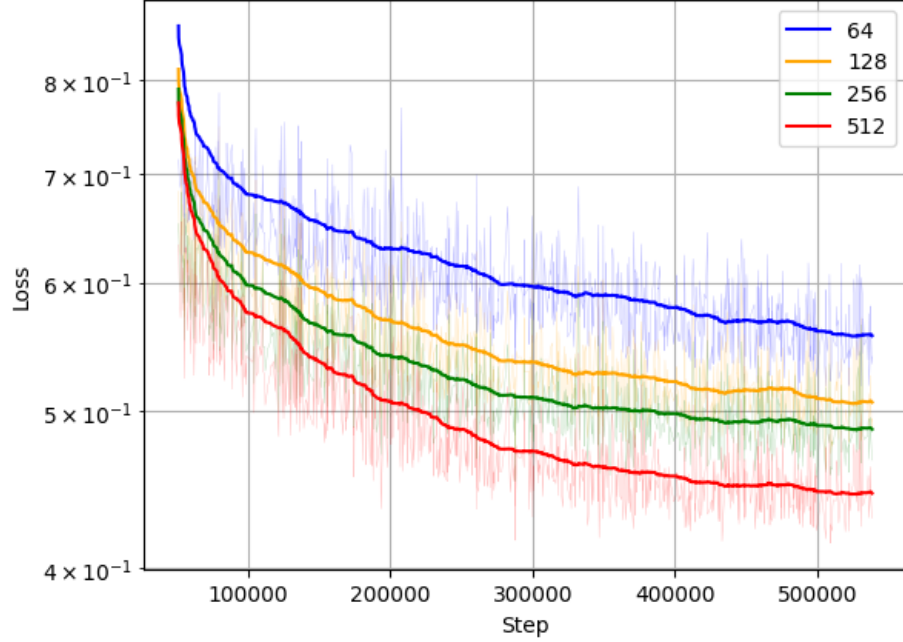


Figure 3.4: The upscaling verification test done on the maximal update parametrization implementation that was eventually used for the scaling law development. The training curves from larger models have lower loss than their smaller counterparts. This indicates a correct implementation. The legend indicates the value of d_{model} for each training loss curve. This is the only difference between models for each curve. Note that these models were trained on the smaller NHANES [1] dataset described in Section 3.1.1, a batch size of 64, and with the incorrect data loader implementation described in Section 3.1.3. Their final loss values are thus not directly comparable to other runs with the full dataset or with a different batch size.

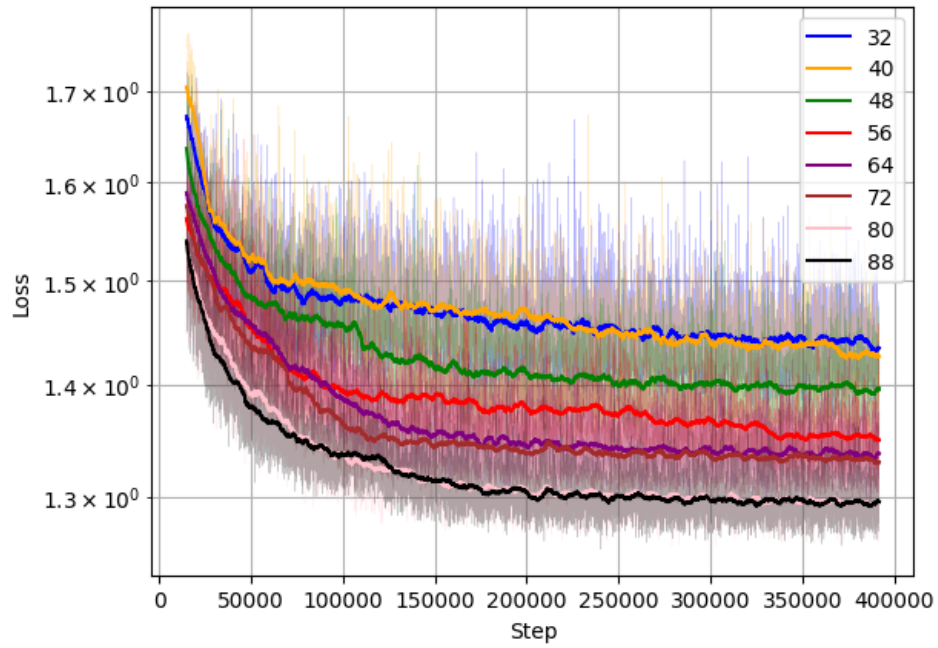


Figure 3.5: Training loss curves for the smallest model sizes used. The legend indicates the value of d_{model} for each training loss curve. This is the only difference between models for each curve. The models are trained on the full [NHANES \[1\]](#) dataset for one epoch with a head size of 8.

Chapter 4

Scaling Law Development

In this Chapter, scaling laws are developed for BIOT [43] using accelerometer and EEG datasets separately. For each dataset, a scaling law is developed using Hoffmann *et al.*'s [16] iso-FLOP curve method and again with Hoffmann *et al.*'s [16] parametric loss function method.

There are two main areas in which choices must be made when using both of these methods to develop a scaling law. One such area is with regards to which model sizes and number of training tokens to use when collecting experimental training curve data. The other is related to the use of the iso-FLOP curve method and involves the decision about what compute budgets to use when generating iso-FLOP curves and which experimental training runs to include for each iso-FLOP curve. Hoffmann *et al.* [16] counsel that the minimum points for each iso-FLOP curve should be relatively clear. To my knowledge, there are no other more deterministic metrics which have been suggested to inform the choices described above. I apply Hoffman *et al.*'s [16] methods with this guidance taken into consideration. However, it is not within the scope of this work to complete a sensitivity analysis on the effects of these choices in order to improve upon or validate the methods which they document in [16]. Rather, the usefulness of each developed scaling law is validated after its development, as is explained next.

Each scaling law is tested by using it to predict the optimal balance of number of parameters D and number of training tokens N for a compute budget C which is 4 to 5 times larger than the largest iso-FLOP curve used to create the scaling law. Then, another model which is 4 times larger than the optimal model is trained with 4 times fewer training tokens. This maintains the same compute budget but alters the balance of D and N such that they are significantly different from the predicted optima. This is done to mimic the

test done by Hoffmann *et al.* [16] when validating their scaling law. However, I also train a model which is 4 times smaller on 4 times as many training tokens. By doing both of these I can compare the performance of the model which is predicted to be optimal to both a larger and smaller model. If the optimal model performs best, I accept it as being sufficiently accurate to be considered useful for researchers attempting to scale up BIOT.

In Section 4.1, a scaling law is developed using the iso-FLOP curve method [16] with the NHANES [1] dataset. It is found not to make accurate predictions. Lessons learned from these experiments are applied to Sections 4.2 and 4.3 where scaling laws are developed with the NHANES [1] dataset with the iso-FLOP curve method [16] and the parametric loss function method [16] respectively. Each scaling law is found to make more accurate predictions than the first attempt based on a successful evaluation by means of the test described above.

In Sections 4.4 and 4.5 scaling laws are developed for BIOT [43] with the EEG dataset using the iso-FLOP method [16] and the parametric loss function method [16] respectively. They are both found to be useful scaling laws based on the application of the same testing methodology used in the other sections.

4.1 Accelerometer–Iso-FLOP Method Attempt 1

In this Section, a scaling law is developed for the NHANES [1] dataset using the iso-FLOP curve method [16]. First, training runs are completed for an array of BIOT [43] models with various values of d_{model} . Then, iso-FLOP curves are developed for a set of distinct compute budgets. From these, a scaling law is developed by fitting a power law to these minima.

The scaling law is then used to predict the optimal balance of number of parameters and number of training steps for a compute budget 4 times larger than that used for the largest iso-FLOP compute budget. The scaling law is tested against a larger and a smaller model with the same compute budget as described above.

It is found that the larger test model performs best, indicating that the scaling law was unable to predict the optimal balance with sufficient accuracy. Lessons learned from the experiments and results of this section are then used in Section 4.2 to improve upon this first scaling law. Note that the tests in this section are done using the incorrect data loader implementation described in Section 3.1.3.

Table 4.1: Breakdown of training runs done in Section 4.1.1. All size ranges are filled with models at increments of 32 and no sizes are repeated. “Full” references the dataset with all 285,922 hours of data while “Small” references a subset of that dataset with only 95,756 hours of data as described in Section 3.1.1.

Size Range (d_{model})	Dataset	Number of Epochs
64	Full	4
96	Full	3
128	Full	2
[160, 224]	Full	1
[256, 512]	Small	1

4.1.1 Training runs

Training runs were done for models with $d_{model} = 64$ to $d_{model} = 512$ inclusively with steps of 32 between since the head size was set to 32. At this point, the batch size used was 64. Learning rate and weight decay were thus 0.0001295 and 0.0002718 as documented in Section 3.3.4. This resulted in 16 training curves. These training runs were originally generated using the smaller version of the NHANES [1] dataset for 1 epoch only. However, early iso-FLOP curve plots showed that the model sizes being studied were likely optimal at higher compute budgets than those being used. Therefore, more training tokens were required in order to reach the compute budget at which some of the model sizes under study were optimal.

Upon observing this, the remainder of the NHANES [1] dataset was downloaded. Then, the models with $d_{model} = 224$ or less were trained again with the expanded dataset. Some were trained for multiple epochs. The full breakdown of which models were trained with which dataset and for how many epochs can be found in Table 4.1.

Moving average smoothing with a window of 100 steps¹ was applied to each training curve in order to reduce noise.

¹Tensorboard [3] was used to save the training loss for each run. By default, the training loss is saved once for each 50 steps. Thus, the moving average window was the average of 100 training loss values across 5000 steps.

4.1.2 Iso-FLOP curves

In order to generate iso-FLOP curves, the approximation:

$$F = 6ND \quad (4.1)$$

from Kaplan *et al.* [20] where F represents the number of FLOPs, N represents the number of training tokens, and D represents the number of learnable parameters in the model was used. By setting F to a constant compute budget, the appropriate step on each training curve where that compute budget is reached can be found. In practice, I replace N with $N_S S$ where N_S is the number of tokens per step and S is the number of steps. Then I set $\frac{F}{6N_S}$ to a constant C and find the corresponding S for each training curve. This simplifies calculations made throughout this chapter by reconfiguring equation 4.1 to:

$$C = DS \quad (4.2)$$

where C represents the compute budget and is proportional to F .

The interval of 32 between values of d_{model} made it difficult to detect a clear minimum from the plotted points. Thus, curves were created by fitting the following equation to the data points of each compute budget:

$$\log L = a(\log D)^2 + b(\log D) + c \quad (4.3)$$

where a , b , and c are constants to be fit. This is the equation of a quadratic fit to the log-transformed loss and number of parameters of each data point. Note that Hoffmann *et al.* [16] also fit a parabola to their points in order to find a precise minimum. The curve is fit using the Levenberg-Marquardt method [27] to minimize the mean square error between the fit and the experimental results. The optimal number of parameters for the compute budget of each iso-FLOP curve is then taken to be the minimum of the fitted log-space quadratic.

Many values for the compute budget C were explored in order to find the most useful iso-FLOP curves for developing a scaling law. This exploration was done purely by inspection to determine for what range of C there existed enough experimental data to get what appeared to be accurate estimates of optimal allocations of S and D . An early set of iso-FLOP curves is shown in Figure 4.1.

This set of iso-FLOP curves has values of C set to between 251 billion and 2,511 billion with values equally spaced on a log scale. The two iso-FLOP curves with the largest values for C have fits that appear significantly different than the others. This

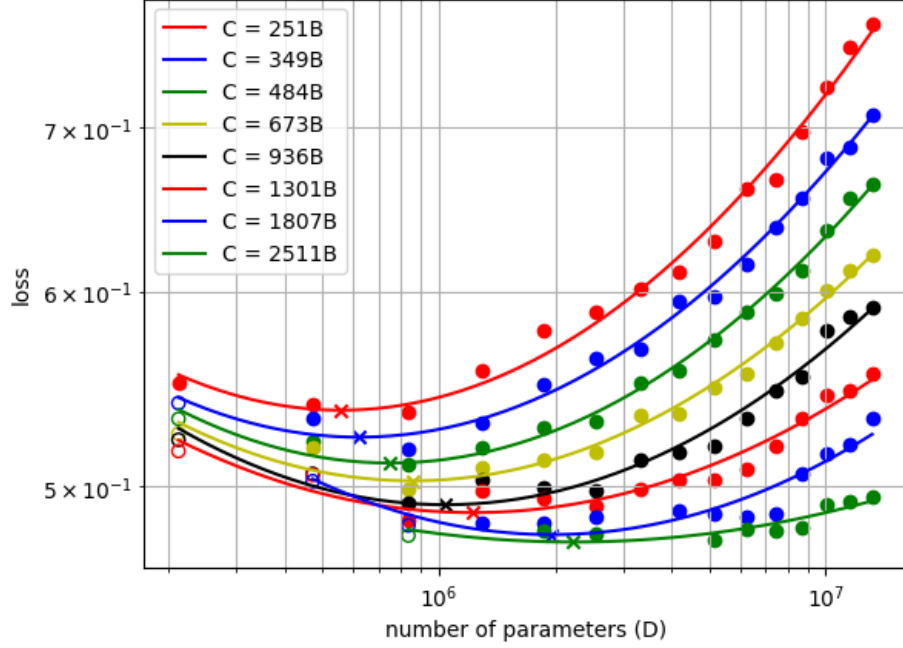


Figure 4.1: Iso-FLOP curves for NHANES [1] data using results from Section 4.1.1 with a wide range of compute budgets. Each point represents the loss and number of parameters from a training curve at a number of steps such that the appropriate compute budget is attained. The points that are filled-in represent a sample taken within the first epoch of a training run while the points which are not filled-in represent a sample taken after the first epoch of a training run. The minimum points are labeled with an x.

is because they surpass the compute budget used to train one or both of the smallest models ($d_{model} = 64, 96$) shown on the plot. That means that no point is available at the corresponding number of parameters to fit an iso-FLOP curve. This inconsistency results in minimum loss points that occur at higher values of D than may be expected by observing and extrapolating from only the iso-FLOP curves with smaller values of C . Furthermore, both of the largest iso-FLOP curves are also missing other points near where the predicted minima are.

It was thus determined that these two larger compute budgets did not have clear enough minima to be included. A smaller range of C was then used to generate new iso-FLOP curves. A range of 351 billion to 1 trillion was chosen. The resulting iso-FLOP curves are shown in Figure 4.2

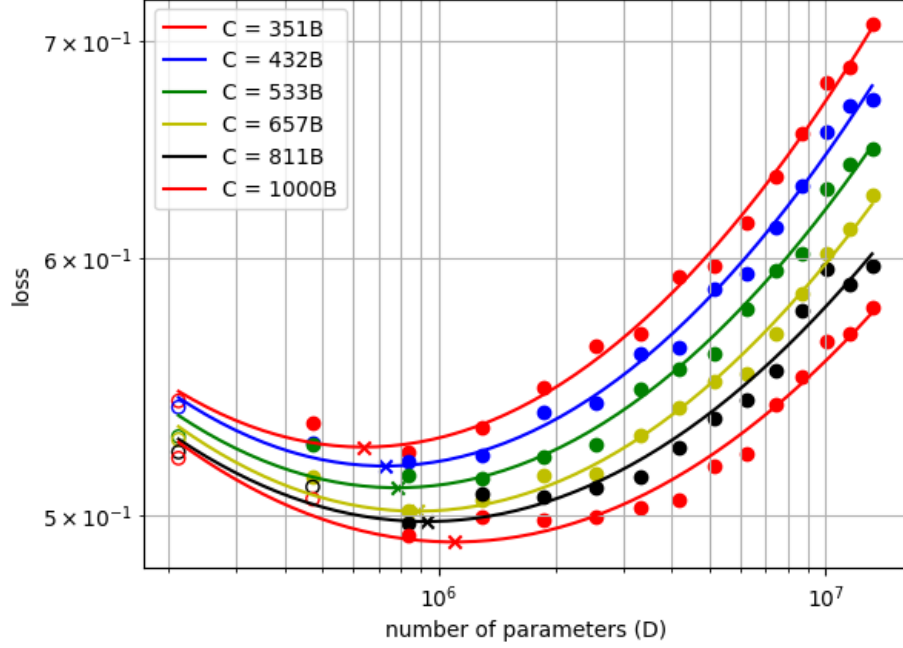


Figure 4.2: Iso-FLOP curves for NHANES [1] data using results from Section 4.1.1 with a reduced range of compute budgets. Each point represents the loss and number of parameters from a training curve at a number of steps such that the appropriate compute budget is attained. The points that are filled-in represent a sample taken within the first epoch of a training run while the points which are not filled-in represent a sample taken after the first epoch of a training run. The minimum points are labeled with an x.

The iso-FLOP curves generated using this range were then used to fit a scaling law to predict the optimal values of D and S for larger compute budgets. This was done by fitting equations 2.3 and 2.4 to the minimum points of the iso-FLOP curves. This fitting was also completed using the Levenberg-Marquardt method [27] to minimize the mean squared error between the fit and the experimental data. The iso-FLOP curves yielded by the reduced range of C values resulted in a scaling law with $D = 1.4133 \times C^{0.48997}$ and $S = 0.69874 \times C^{0.51051}$.

This suggests that both number of parameters and number of steps should be scaled about equally when increasing compute budget by some multiplicative factor. For example, if the compute budget is increased by a multiple of 4, the number of steps and the number of parameters should each be increased approximately by a multiple of 2. This is the same

Table 4.2: The results from the test of the scaling law developed in Section 4.1. The final loss is the average loss of the last 100 recorded steps of training.

Model width d_{model}	Number of parameters D	Number of steps S	Final loss L
96	472,611	8,463,620	0.5009
224	2,536,354	1,577,066	0.4734
448	10,090,307	396,420	0.4670

result obtained by Hoffmann *et al.* [16] for language transformers.

4.1.3 Scaling law testing

The chosen scaling law was tested by using it to predict the optimal D and S for a compute budget 4 times larger than the largest iso-FLOP curve used to create the scaling law. Since that highest compute budget was $C = 10^{12}$, $C = 4 \times 10^{12}$ was chosen. By using this value in equation 2.4 with $a_d = 1.41333089$ and $b_d = 0.48996819$, the predicted D_{opt} was determined to be approximately 2,112,770.

Given that models in this section had a head size of 32, a model with a value of d_{model} which was divisible by 32 had to be chosen to be the optimal test model. Thus, a model with $d_{model} = 224$ and $D = 2,536,355$ was chosen. This was approximately 20% larger than the predicted optimum.

Since the predicted D_{opt} could not be matched exactly, the number of steps S was chosen such that the compute budget $C = 4 \times 10^{12}$ was as closely maintained as possible. This value was 1,577,066 steps. Then, a model that had approximately 4 times as many parameters and another which had approximately 4 times fewer parameters were also selected and trained for a number of steps that maintained the compute budget of 4 trillion as closely as possible. A breakdown of all 3 models used for testing can be found in Table 4.2 and a plot of the 3 training runs can be found in Figure 4.3.

The model with the best performance was the largest model with $d_{model} = 448$. This indicates that the scaling law was unsuccessful in predicting the best values for D and S with enough accuracy to be deemed useful.

4.1.4 Discussion

In this section, a scaling law that could accurately predict the optimal balance of training steps and model parameters for a 4 times increase in compute budget was not successfully

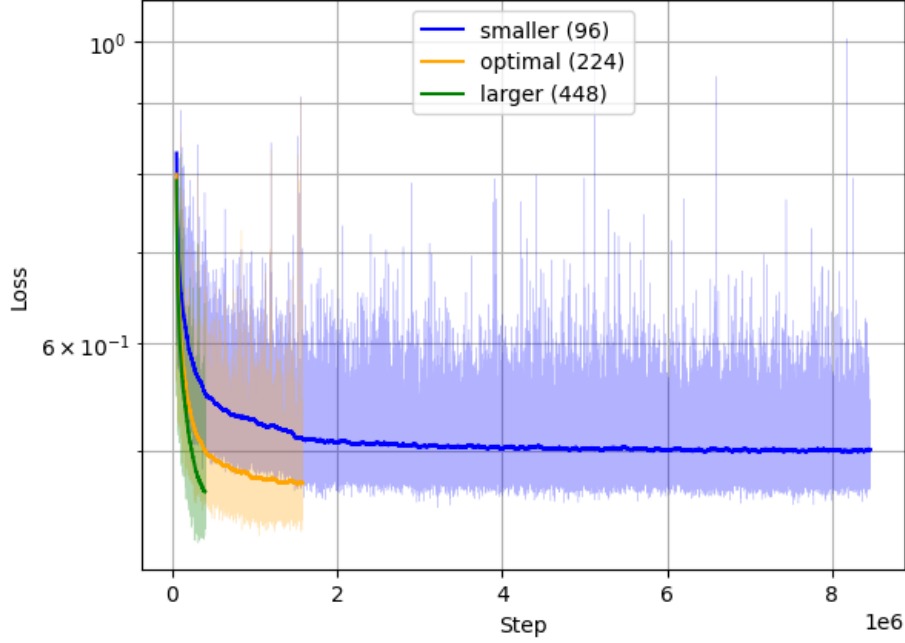


Figure 4.3: Training curves to test the first iteration of the accelerometer scaling law.

discovered. Note that since the larger test model was the one with the lowest loss, the scaling law underestimated the optimal number of parameters for the given compute budget. Thus, the fact that the optimal test model was 20% larger than the predicted optimal model was an advantage, but an insufficient one.

It is concluded from this result and from visualizing the iso-FLOP curves that the obtained minima were likely not sufficiently accurate. This was mostly due to the sparse distribution of data points around the presumed minima. Furthermore, some noise seems to be present between models of different sizes as was observed in Section 3.2.4. The following section attempts to address the challenge of obtaining clear minima for a set of iso-FLOP curves by decreasing the head size such that more models of sizes that closely surround the presumed optimum points can be trained and by training some smaller models more than once with different seeds.

Note again that the results in this section were obtained using the incorrect implementation of the data loader described in Section 3.1.3. The results are left here despite this and despite the failure to generate an accurate scaling law in order to motivate the decision to reduce the head size to 8 in the next section, to show concretely some iso-FLOP curves

which do not show significantly clear minima, and to show an example of a scaling law test which indicates an inaccurate scaling law.

4.2 Accelerometer–Iso-FLOP Method Attempt 2

In this section, changes are made to the previously employed process in an attempt to improve upon the results. Specifically, the challenge of unclear iso-FLOP minima is addressed. First, the head size is changed from 32 to 8. This allows a larger number of choices for d_{model} when collecting experimental results, especially at smaller model sizes. Then, iso-FLOP curves are fit only to points that are near the presumed optimum rather than to all available points. A scaling law is then generated in the same way as before, by fitting a power law to the observed optimum points for each iso-FLOP curve.

Upon testing this scaling law, it is found that the model it predicts to be optimal at a higher compute budget outperforms the larger and smaller models it is tested against. This suggests that the changes made in this section were useful for developing an accurate scaling law. Other conclusions are then discussed in more detail.

4.2.1 Training runs

In addition to reducing the head size from 32 to 8, the batch size was also increased from 64 to 256. This was done since it became clear from the first round of tests and scaling law exploration that optimal models would have d_{model} much smaller than 2048 for the compute budgets under study. Therefore, a larger batch size could be used without the need for splitting batches during training on available hardware.

The base learning rate and weight decay were thus also changed to accommodate the new head and batch sizes. As documented in Section 3.3.4, the base learning rate used was 0.001132 while the weight decay was 0.00000371.

All the training runs done with these new hyperparameters are documented in Table 4.3. The smaller head size allows the smallest models to have values for d_{model} separated by only 8. This could have been done for all models trained but was not deemed necessary for larger models which had values of d_{model} separated by increments of 16. Furthermore, some smaller models were trained more than once with different random seeds to increase the variety in that range of model sizes. This along with the smaller increment was done in an attempt to counteract the high levels of noise for small models seen in the previous

Table 4.3: The training runs done for Section 4.2.1. “Interval” describes the change in d_{model} for neighboring models in the same range.

Size Range (d_{model})	Interval	Number of Seeds
[32, 128]	8	2
[144, 384]	16	1
512	N/A	1

attempt such that minimum points for each iso-FLOP curve could be determined with more accuracy.

4.2.2 Iso-FLOP curves

As was done in Section 4.1.2, iso-FLOP curves were generated by first sampling the smoothed loss of each training curve at a range of compute budgets. For this attempt, the chosen compute budgets were accepted as having clear enough minima only if the range of D which was presumed to hold the true minimum was populated with significantly more points than in the previous attempt. Then, some samples which were likely not in the neighborhood of the presumed optimal points were discounted and not used when fitting an iso-FLOP curve. This was done by finding the point with the minimum loss and then excluding any points with loss higher than that loss by more than 0.1. This mostly removed points from models which were much larger than the likely optimal model for each iso-FLOP curve. The value of 0.1 was chosen empirically with the goal of removing outliers only. Note that Hoffmann *et al.* [16] also only use points in the neighborhood of the presumed minima. Although they present no heuristic for choosing those points other than to assert that they ensure a clear minimum is available [16].

Then, for each compute budget, equation 4.3 was fit to the corresponding remaining points in order to generate an iso-FLOP curve. The resulting iso-FLOP curves can be seen in Figure 4.4.

The optima of each set of iso-FLOP points are not indisputably clear due to noise. Although, the increased number of smaller models trained was successful in more effectively surrounding each apparent minimum with data points. By again fitting equation 4.3 to each set of iso-FLOP points in order to generate iso-FLOP curves, it was theorized that the resulting minima would be more accurate due to the additional neighboring data points and the exclusion of distant points.

Note that since the batch size used here was 256 rather than 64, each step includes 4

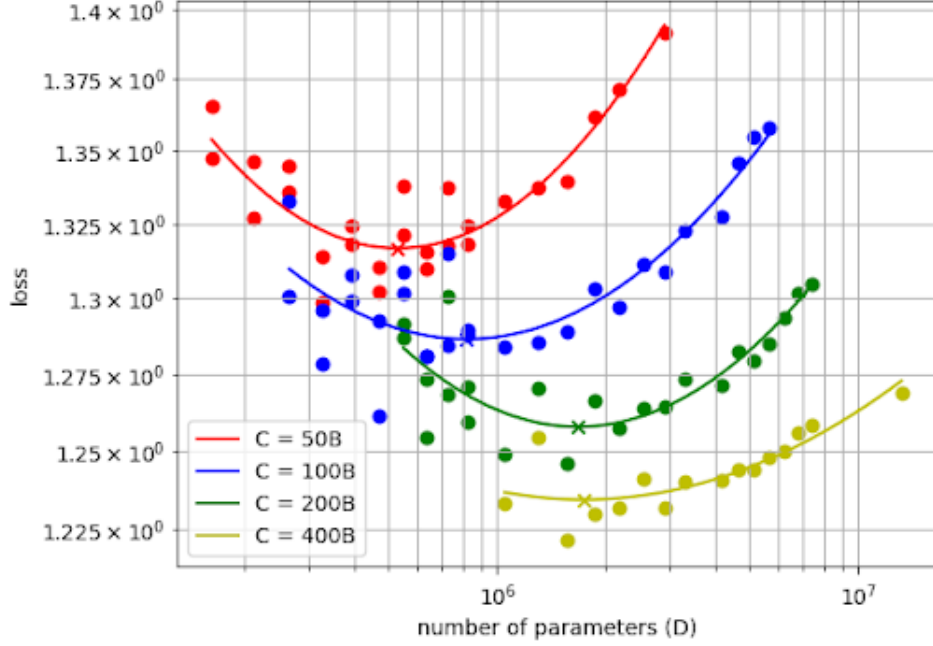


Figure 4.4: The iso-FLOP curves generated in Section 4.2.2. The points shown are those used to fit the iso-FLOP curves. The minimum points are labeled with an x.

times as many tokens. Since I defined $C = DS$, one unit of compute budget here is 4 times larger than a unit of compute budget in Section 4.1. The increased batch size also increases the loss as discussed in Section 3.3.3. Thus it is not useful to compare loss between each section directly, but insights can still be informed by comparing relative difference across iso-FLOP curves between this section and the last.

The resulting law was defined by $D = 2.6436 \times C^{0.50433}$ and $S = 0.3782 \times C^{0.49567}$. The exponents here suggest that the number of parameters in a model D should be increased by an approximately equal factor to the number of steps S for a given increase in compute budget C . This is similar the previously developed scaling law, which also suggested that D and S should be scaled equally. However, the law is shifted such that predictions for the optimal number of parameters will be nearly 2 times larger for any given compute budget. Since the test of the last scaling law showed that the larger model performed better than the model that was predicted to be optimal, this is a positive sign.

Table 4.4: The results from the test of the scaling law developed in Section 4.2. The final loss is the average loss of the last 100 recorded steps of training. Each model size has two final loss scores from two different random seeds followed by the average of the two scores.

Model width d_{model}	Number of parameters D	Number of steps S	Final loss L
144	1,049,331	1,905,976	1.2081
			1.1991
		Mean Loss:	1.2036
288	4,172,259	479,356	1.1860
			1.2021
		Mean Loss:	1.1940
576	16,638,915	120,200	1.2054
			1.1977
		Mean Loss:	1.2016

4.2.3 Scaling law testing

The scaling law was tested with the same procedure which was used for the previous scaling law save that the test was repeated twice with different random seeds due to the noise observed in the iso-FLOP plot in Figure 4.4. Additionally, since a more optimal batch size² was being used, C for the test was chosen to be 5 times larger than the largest iso-FLOP curve rather than only 4 times. This also helps to ensure that this scaling law is not at an advantage due to differences in the chosen values of C for the iso-FLOPs in this attempt versus the last.

In this case, the predicted optimal number of model parameters D_{opt} for $C = 2 \times 10^{12}$ was 4,227,747. The model closest in size to that while maintaining a head size of 8 had $d_{model} = 288$ and $D = 4,172,259$. The number of steps used with this model size to maintain a compute budget of $C = 2 \times 10^{12}$ was 479,356. A model with approximately 4 times as many parameters and another with approximately 4 times fewer parameters were also trained using the same compute budget. The full list of models used to test the scaling law and the results for each can be found in Table 4.4. Figure 4.5 shows the training curves for this test.

The model with the best average performance was the model which the scaling law predicted would be optimal with $d_{model} = 288$. In this case, the scaling law was successful in predicting optimal values for D and S .

²Optimal in that it used the available GPU more effectively.

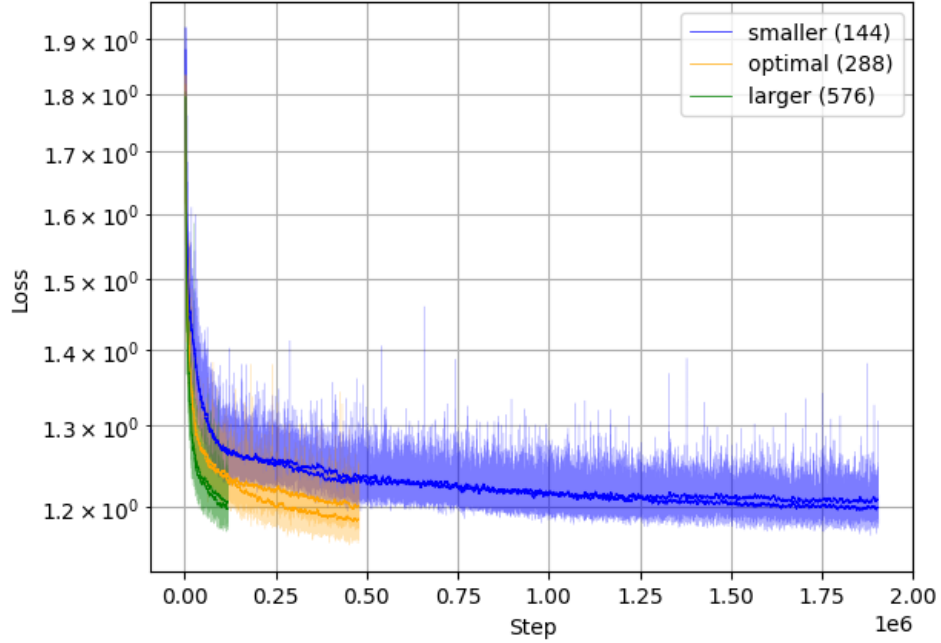


Figure 4.5: Training curves to test the accelerometer scaling law developed in Section 4.2

4.2.4 Discussion

In this section, a scaling law was developed which did accurately predict an optimal balance of number of parameters and number of steps for training a BIOT [43] model at an increased compute budget. This suggests that the efforts to improve the accuracy of the fitted iso-FLOP curves in determining the optimal points for each sampled compute budget were successful. These efforts included decreasing the head size such that more variety in model sizes around the presumed optima was possible, training smaller models more than once with different random seeds, and discounting points that were significantly far from the presumed optima when fitting the iso-FLOP curves.

4.3 Accelerometer–Parametric Loss Function Method

In this section, a parametric loss function is fit to the experimental data gathered in Section 4.2.1. As in Hoffmann *et al.* [16], the parametric loss function is then used directly to find

Table 4.5: Results from fitting equation 4.4 to the experimental training loss curves from Section 4.2.1

Constant	E	α	β	S_C	D_C
Value	1.038	0.4205	0.3357	673.9	2354

optimal numbers of parameters for various compute budgets rather than fitting iso-FLOP curves. These minima are then used to fit a power law as done with minima from iso-FLOP curves.

Upon testing this scaling law, it is found that the model which is predicted to be optimal at a higher compute budget outperforms the larger and smaller models it is tested against.

4.3.1 Parametric loss function

Rather than use equation 2.5 directly, that equation is modified to be given the following arrangement:

$$L = \left(\frac{S_C}{S}\right)^\alpha + \left(\frac{D_C}{D}\right)^\beta + E \quad (4.4)$$

where N and N_C have been replaced with S and S_C following the simplification made in Section 4.1.2. Additionally, the constants α and β are applied to the numerator and denominator of their respective terms rather than only to the denominator. Since the numerator is a constant and both α and β are constants, no change in the space of possible resulting fits has been made.

Equation 4.4 was fit to all the training curves using the Nedler-Mead [13] method to minimize the mean squared error of the experimental training loss with the fit. The results for each constant are shown in Table 4.5.

4.3.2 Iso-FLOP curves

Iso-FLOP curves can then be generated using this parametric loss function rather than the noisy experimental data. This is done by setting $DS = C$ for some values of C in the same range as in the previous experiment and then sampling the loss function at various values of D . The resulting iso-FLOP curves can be seen in Figure 4.6.

Note that it is not necessary to generate iso-FLOP curves when using this method. However, they are generated here anyways to allow easier visual comparison between this method and the last.

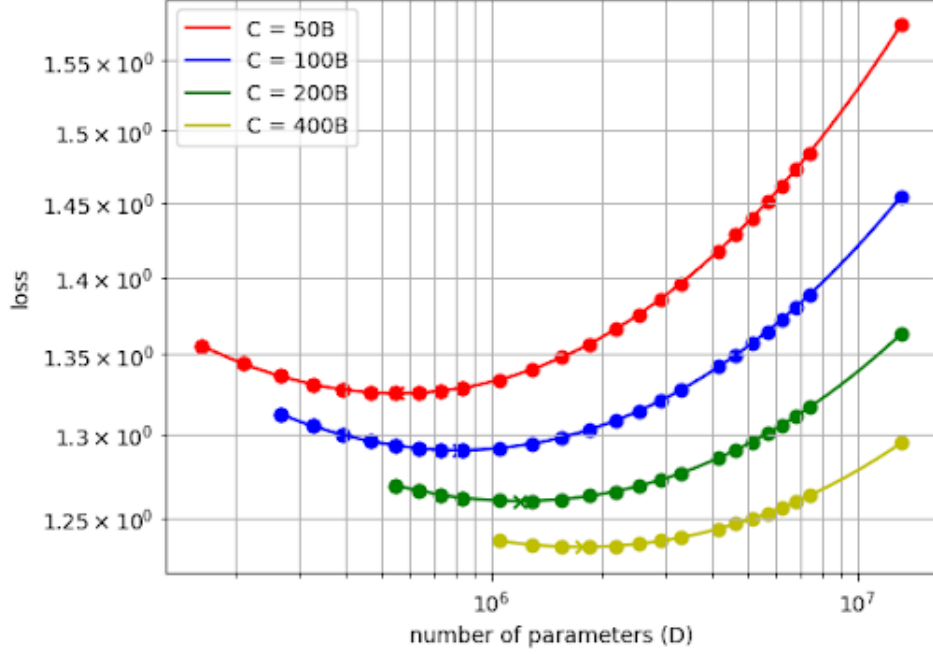


Figure 4.6: The iso-FLOP curves generated using the parametric loss function fitted on accelerometer training curve data from Section 4.2.1. Points represent loss estimations using the parametric loss function at the corresponding D and S such that C is as shown in the legend. Points which line up with models that were trained in Section 4.2.1 are used to show their spread. However, the parametric loss function is used directly to draw the curve rather than fitting a quadratic to the plotted points.

A scaling law was generated by numerically minimizing the fitted loss function under the constraint $DS = C$ for the values of C shown in Figure 4.6. The resulting law was defined by $D = 0.62369 \times C^{0.55603}$ and $S = 1.6034 \times C^{0.44396}$. This is similar to the previously developed scaling law but different enough that its prediction for the optimal balance of number of parameters and number of steps for a compute budget of $C = 2 \times 10^{12}$ was different than what was predicted by the previous scaling law. Thus, another test was conducted to confirm the accuracy of the scaling law.

Table 4.6: The results from the test of the scaling law developed in Section 4.3. The final loss is the average loss of the last 100 recorded steps of training. Each model size has two final loss scores from two different random seeds followed by the average of the two scores.

Model width d_{model}	Number of parameters D	Number of steps S	Final loss L
152	1,168,427	1,711,702	1.2192
			1.2124
			Mean Loss: 1.2158
296	4,406,555	453,869	1.1913
			1.1894
			Mean Loss: 1.1903
592	17,574,707	113,799	1.2068
			1.2012
			Mean Loss: 1.2040

4.3.3 Scaling law testing

The scaling law was tested with the same procedure and the same target compute budget of $C = 2 \times 10^{12}$ as was used for the previous method in Section 4.2.3

In this case, the predicted optimal number of model parameters D_{opt} for $C = 2 \times 10^{12}$ was 4,313,131. The model closest in size to that while maintaining a head size of 8 had $d_{model} = 296$ and $D = 4,406,555$. The number of steps used with this model size to maintain a compute budget of $C = 2 \times 10^{12}$ was 453,869. A model with approximately 4 times as many parameters and another with approximately 4 times fewer parameters were also trained using the same compute budget. The full list of models used to test the scaling law and the results for each can be found in Table 4.6. Figure 4.7 shows the training curves for this test.

The model with the best average performance was the model which the scaling predicted would be optimal with $d_{model} = 296$. In this case, the scaling law was successful in predicting optimal values for D and S .

4.3.4 Discussion

In this section, a scaling law was developed, using the parametric loss function method [16], which did accurately predict an optimal balance of number of parameters and number of steps for training a BIOT [43] model at an increased compute budget. This scaling law

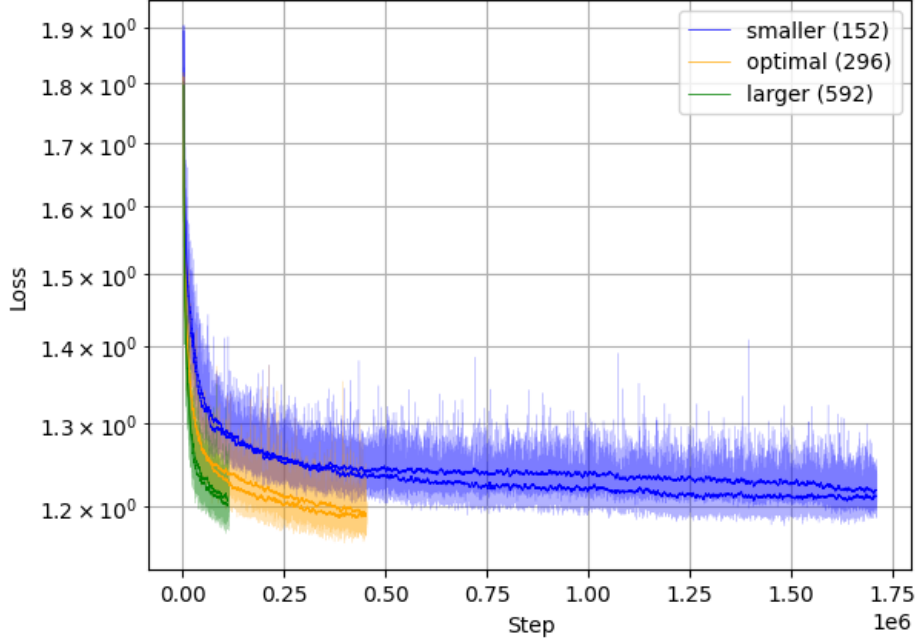


Figure 4.7: Training curves to test the accelerometer scaling law developed with the parametric loss function method.

suggests that the number of parameters should be scaled proportionally to compute budget to the power of 0.56 ($D \propto C^{0.56}$) whereas the previous section’s scaling law which was developed with the iso-FLOP curve method [16] recommended $D \propto C^{0.50}$. This difference is of a slightly larger size compared to the difference between scaling laws reported by Hoffmann *et al.* [16] using these same methods.³

4.4 EEG–Iso-FLOP Method

In this section, the iso-FLOP method [16] is applied to create a scaling law using the TUH-EEG [28] dataset. Lessons learned from the first attempt at applying this method to the NHANES [1] dataset are applied here as well. Specifically, the head size of the models used is maintained at 8 such that a sufficient spread of model sizes around the presumed minima of a sufficient number of compute budgets is available. Additionally, samples from training

³See Section 2.5 for the values that they report.

curves which are significantly sub-optimal for a given compute budget are excluded from iso-FLOP curve fitting for that compute budget as was done in Section 4.2.2.

Yet again, the resulting scaling law is successfully able to predict an optimal balance of number of parameters and number of steps to outperform one larger and one smaller model all at the same elevated compute budget. Conclusions related to this result are then presented and discussed.

4.4.1 Training runs

The head size for BIOT [43] trained on EEG data was maintained at 8 in order to continue to allow the same granularity of differences in model sizes for low values of d_{model} . However, the batch size was reduced to 100. This was done because the number of channels in the EEG data was 24 which significantly increased the sequence length being passed to the model since for both accelerometer and EEG data, an entire 10 seconds are used in each sample. Recall that the network splits input into 1 second tokens with 50% overlap resulting in 19 tokens for each channel. The sequence length is the number of tokens which is thus 8 times more for EEG given the 8 times more channels. Note that the maximum model size for the EEG data without needing to split a batch was still lower than for the accelerometer data even with the decreased batch size. But, given the additional tests done in Sections 4.2 and 4.3, it was predicted that a batch size of 100 would still allow for large enough models to be trained in order to develop and test a scaling law.

Since a new dataset was being used, another Optuna [5] hyperparameter study was done to determine what base learning rate and weight decay should be used. As documented in Section 3.3.4, the base learning rate was 0.0000945 and the weight decay was 0.00000317.

All the training runs done with the TUH-EEG [28] dataset are documented in Table 4.7. Due to the lower number of hours in this dataset compared to NHANES [1], the models are trained for 2, 3 or 11 epochs rather than just 1. This was done after initial tests showed that the presumed optima at various compute budgets were not sufficiently defined when plotting iso-FLOP curves. However, none are trained more than once with different seeds since relative noise was not deemed to be as intense as for the NHANES experiments.

Table 4.7: The training runs done for the development of a scaling law with EEG data. Interval refers to the difference in values of d_{model} for neighboring models in the same range.

Size Range (d_{model})	Interval	Epochs
[32, 96]	8	11
[104, 128]	8	3
[136, 384]	16	2
448	N/A	2
512	N/A	2

4.4.2 Iso-FLOP curves

As was done in Section 4.2.2, iso-FLOP curves were generated by first sampling the smoothed loss of each training curve at a range of compute budgets. Then, some samples which were likely not in the neighborhood of the presumed optimal points were discounted. This was done by finding the point with the minimum loss and then excluding any points with loss higher than that loss by more than 0.1. This effectively balanced the available points such that there were not significantly more points representing models larger than optimal than there were points representing models smaller than optimal for most compute budgets. Then, for each compute budget, equation 4.3 was fit to the corresponding remaining points in order to generate an iso-FLOP curve with a clear minimum. The resulting iso-FLOP curves can be seen in Figure 4.8

The scaling law obtained from these iso-FLOPs was defined by $D = 0.5146 \times C^{0.55598}$ and $S = 1.8495 \times C^{0.44606}$.

4.4.3 Scaling law testing

The scaling law was then tested with the previously established method. The test compute budget was chosen to be 500 billion which is 5 times larger than the largest iso-FLOP curve used to generate the scaling law. For this compute budget, the scaling law predicted that 1,643,789 parameters would be the optimal number. Thus, a model with $d_{model} = 176$ and 1,583,320 parameters was chosen to be the optimal model for the test. The results of the test along with documentation of the number of parameters and training steps for each model used in the test can be found in Table 4.8 while a plot of the training curves can be found in Figure 4.9. Only one instance of each model was trained due to the lower noise observed in the iso-FLOP plot compared to the iso-FLOP plot generated with the NHANES [1] dataset in Figure 4.4.

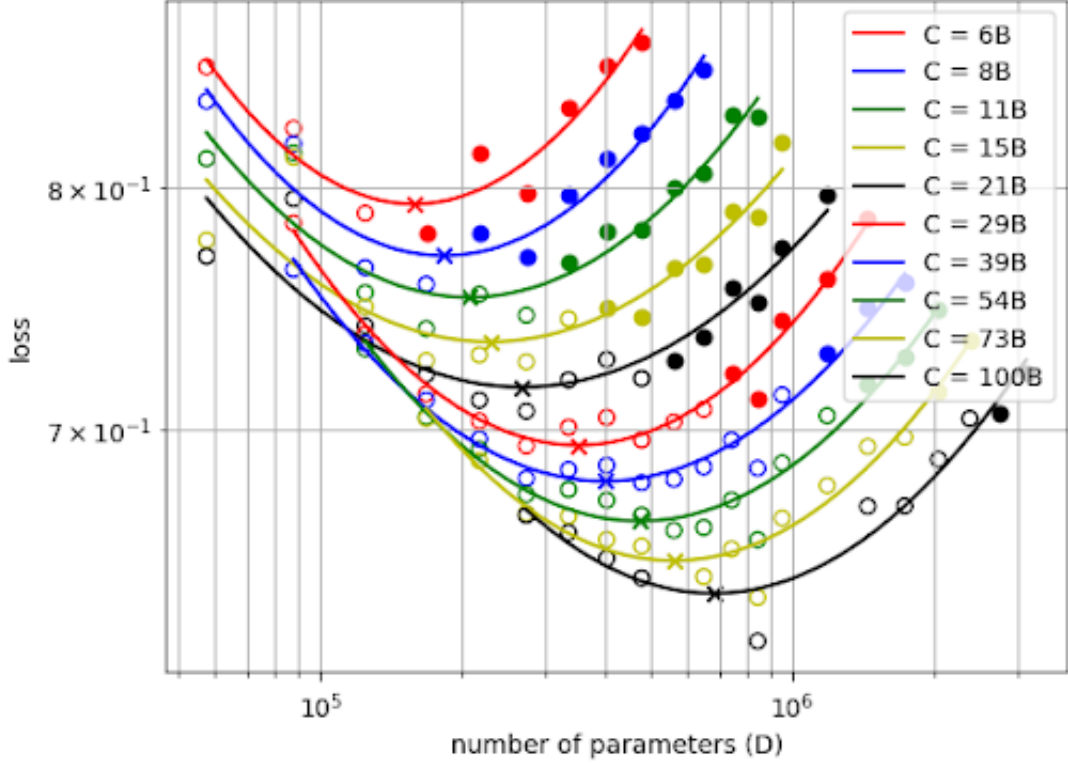


Figure 4.8: The iso-FLOPs using the EEG training curve data. Only points which are used to fit the iso-FLOP curves are shown. Unfilled points indicate samples taken after the first epoch of a training run.

The model with the best performance was the model which the scaling law predicted to be optimal with $d_{model} = 176$.

4.4.4 Discussion

In this section, the iso-FLOP method [16] successfully lead to a scaling law that was able to predict an optimal balance of number of parameters and number of training steps which performed better than both a larger and a smaller model for a compute budget five times larger than the largest used to develop the scaling law.

Specifically, this scaling law recommends $D \propto C^{0.56}$ which is in the same range as

Table 4.8: The results from the test of the EEG scaling law developed with the iso-FLOP curve method. The final loss is the average loss of the last 100 recorded steps of training

Model width d_{model}	Number of parameters D	Number of steps S	Final loss L
88	404,472	1,236,179	0.6283
176	1,583,320	315,792	0.5951
360	6,550,584	76,329	0.6386

the scaling laws developed with the NHANES [1] dataset. Although this result does not confirm that BIOT [43] should be scaled similarly for all types of biosignal data, it does suggest that it should be scaled similarly for EEG data and accelerometer data.

4.5 EEG–Parametric Loss Function Method

In this section, a scaling law for BIOT [43] trained on the TUH-EEG [28] dataset is developed using the parametric loss function approach.

Since no changes to any model hyperparameters such as head size were deemed to be necessary, the training runs done in Section 4.4.1 and documented in Table 4.7 were used again for the development of this scaling law.

The resulting scaling law suggests that the number of parameters should be scaled less than the previous scaling law developed with the iso-FLOP curve method [16] and less than either scaling law developed with the NHANES [1] dataset. The scaling law then successfully predicts a pair of values for D and S for a larger compute budget which outperform a smaller and larger model trained with the same compute budget. Conclusions related to this result are then presented and discussed.

4.5.1 Parametric loss function

As was done in Section 4.3.1, equation 4.4 was fit to the collected experimental data using the Nelder-Mead [13] method. The results for each constant are shown in Table 4.9.

4.5.2 Iso-FLOP curves

Next, iso-FLOP curves were generated by setting $DS = C$ for multiple C s and then sampling for a range of D values. The resulting curves with points showing the sizes of

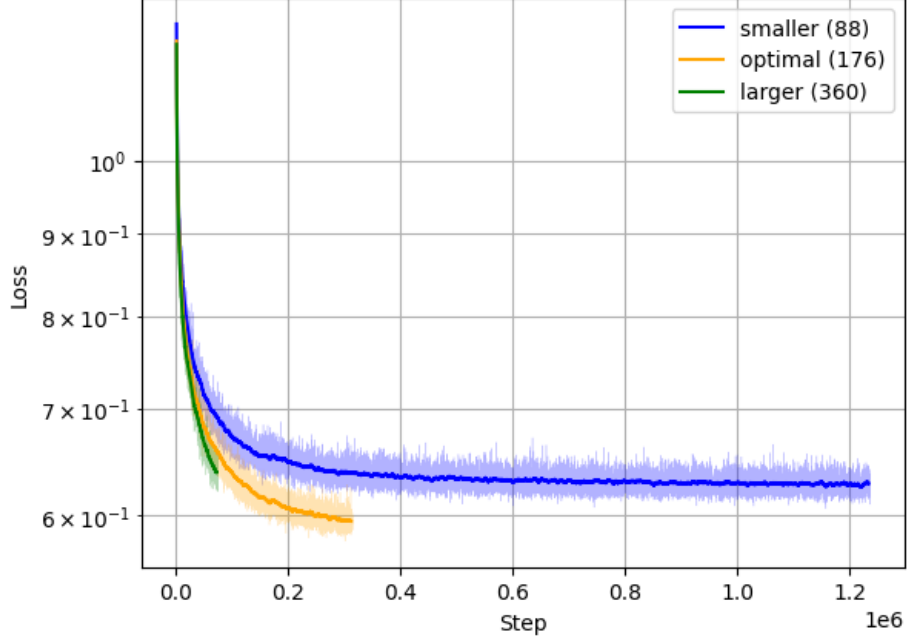


Figure 4.9: Training curves to test the EEG scaling law developed with the iso-FLOP curve only approach. The numbers in the legend indicate the corresponding values of d_{model}

models trained in Section 4.4.1 are shown in Figure 4.10.

The scaling law obtained by minimizing the fitted parametric loss function at each chosen compute budget and then fitting a power law to the discovered minima was defined by $D = 9.5985 \times C^{0.43462}$ and $S = 0.1042 \times C^{0.56538}$. This was significantly different than the previously obtained scaling law and so a test was applied to it to establish its accuracy.

Table 4.9: Results from fitting equation 4.4 to the experimental training loss curves from Section 4.4.1

Constant	E	α	β	S_C	D_C
Value	0.4837	0.4047	0.5265	510.1	3997

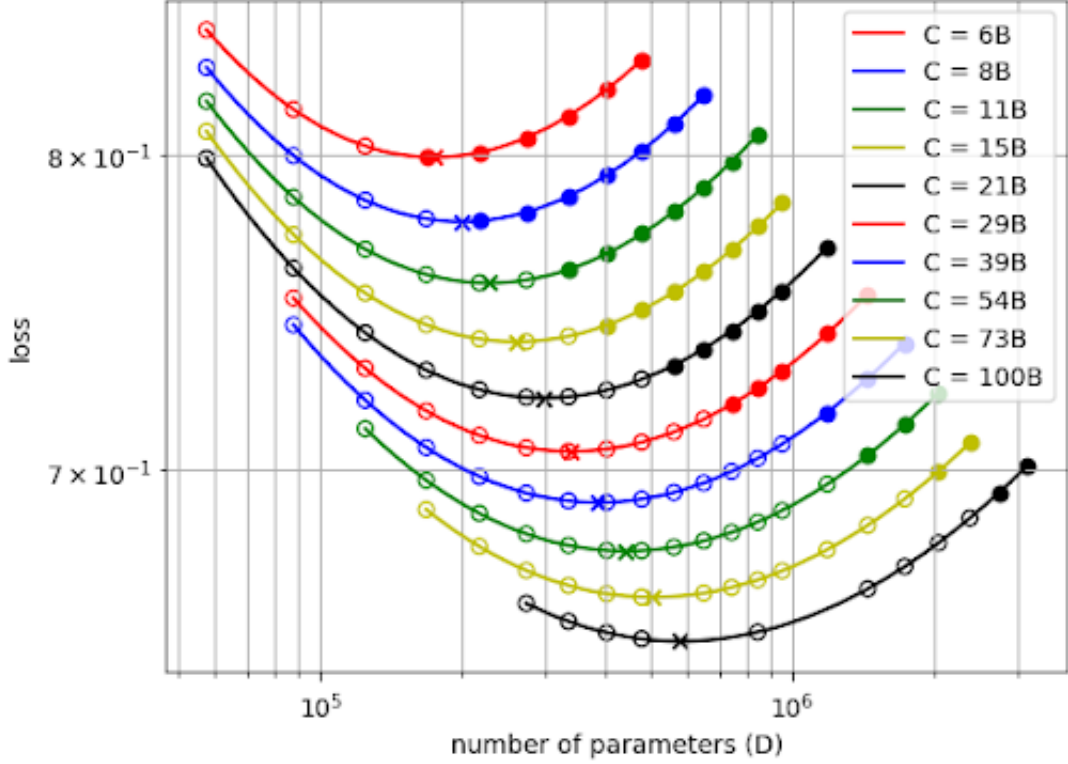


Figure 4.10: The iso-FLOPs using the parametric loss function fitted on EEG training curve data. The points represent loss estimated by the parametric loss function for numbers of parameters D which match models that were used for iso-FLOP curve fitting in Section 4.4.2.

4.5.3 Scaling law testing

The scaling law was tested with the established method used for both the scaling laws developed with the NHANES [1] dataset and the previously developed scaling law with the TUH-EEG [28] dataset. A compute budget of 500 billion was maintained here from the previous EEG scaling law test. The predicted optimal number of parameters at that compute budget was 1,166,253. Thus the d_{model} chosen for testing was 152 which resulted in a model with 1,185,016 parameters. The results for this model as well as the neighboring larger and smaller models can be seen in Table 4.10. Training curves from these tests can be seen in Figure 4.11.

Table 4.10: The results from the test of the EEG scaling law developed with the parametric loss function approach. The final loss is the average loss of the last 100 recorded steps of training

Model width d_{model}	Number of parameters D	Number of steps S	Final loss L
72	273,336	1,829,250	0.6603
152	1,185,016	421,935	0.5850
304	4,680,408	106,828	0.6152

The optimal model with $d_{model} = 280$ has the lowest loss between the three models. This indicates that the scaling law is sufficiently accurate.

4.5.4 Discussion

Both the EEG scaling laws developed with the iso-FLOP curve approach [16] and the parametric loss function approach [16] were able to successfully pass the testing method used to evaluate them. However, they have more significant differences between them than the scaling laws developed for the NHANES [1] dataset do or the scaling laws developed for language transformers by Hoffmann *et al.* [16] do. The scaling law developed in this section recommends $D \propto C^{0.43}$ while the scaling law developed for EEG data using the iso-FLOP curve approach [16] recommends $D \propto C^{0.56}$. It is to be expected that a difference such as this one could occur given that the compute budget used in this thesis to develop the scaling laws was much lower than what was available to Hoffmann *et al.* [16] when developing theirs. Not only does this mean that fewer models could be trained when collecting experimental training curve data, but also the models trained in this phase were smaller which resulted in more noise between different training runs than was observed by Hoffmann *et al.* [16].

Some additional analysis in Appendix B shows a visualization of the tests done to verify the accuracy of the EEG scaling laws developed in this section and the last.

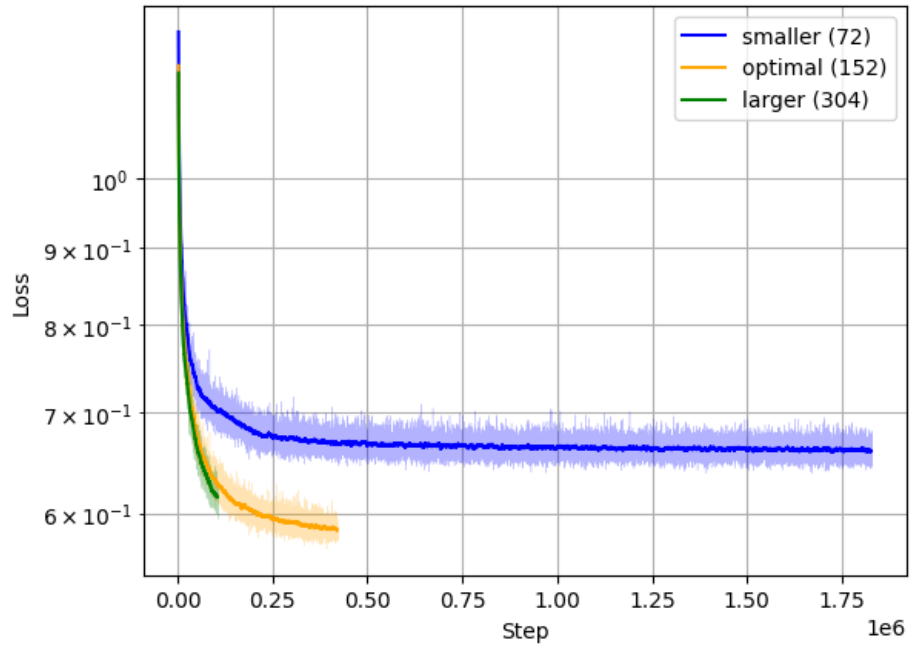


Figure 4.11: Training curves to test the [EEG](#) scaling law developed with the parametric loss function approach. The numbers in the legend indicate the corresponding values of d_{model} .

Chapter 5

Conclusion

In this thesis, scaling laws which predict the balance of number of parameters D and number of training tokens N for a given compute budget C were developed for BIOT [43] with both accelerometer and EEG data to minimize training loss for the unsupervised pre-training task used by [43]. This was done by adopting the commonly used power law model of compute-optimal scaling laws [20, 16], described by equations 2.3 and 2.4. Then, two methods used by Hoffman *et al.* [16] to develop such scaling laws for language transformers were used in this work to fit the power law model for BIOT. These methods were the iso-FLOP curve method [16] and the parametric loss function method [16]. Each method was applied to both BIOT trained with accelerometer data and BIOT trained with EEG data. The resulting scaling laws were tested similarly to the testing method employed by Hoffman *et al.* [16] in their work. Each law was used to predict an optimal model for a compute budget 4-5 times larger than what was used to create the scaling law. Then, models with 4 times as many parameters as were predicted to be optimal and models with 4 times fewer parameters than were predicted to be optimal were also trained with the same compute budget. The final losses of each model were compared and if the optimal model performed best, the scaling law was accepted as sufficiently accurate to be useful for researchers intending to scale up BIOT [43] with a limited compute budget.

In Section 4.1, the scaling law developed for accelerometer data with the iso-FLOP curve method [16] did not pass this test. Adjustments were made to collect more useful experimental pre-training curve data which led to a scaling law in Section 4.2, developed for accelerometer data with the iso-FLOP curve method [16], which did pass the test. Using similar sets of experimental training curve data, a scaling law was successfully developed for accelerometer data using the parametric loss function method [16], and both methods

Table 5.1: The resulting scaling laws from Chapter 4.

Dataset	Method	Scaling Law
Accelerometer	Iso-FLOP	$D \propto C^{0.50}$
Accelerometer	Parametric loss function	$D \propto C^{0.56}$
EEG	Iso-FLOP	$D \propto C^{0.56}$
EEG	Parametric loss function	$D \propto C^{0.43}$

also had success in developing scaling laws for EEG data. The resulting scaling laws can be seen in Table 5.1.

Only the rates of scaling for number of parameters D are shown since the rates of scaling for number of training tokens N or number of steps S is always the complement of the rate of scaling for D . For example, $D \propto C^{0.56}$ implies $N \propto C^{0.44}$ since $C \propto DN$. In practice, researchers should use the scaling laws in Table 5.1 to predict an optimal number of parameters. They should then find a model with a shape such that that number of parameters is as closely achieved as possible. Then, they can find the corresponding number of training tokens or steps based on their compute budget and the resulting size of their model.

5.1 Limitations

In this thesis, compute resources limited the maximum compute budgets and number of models which could be used to test the obtained scaling laws. This led to compute budget increases of 4-5 times compared to the 100 times increase used by Hoffman *et al.* [16]. Thus, the scaling laws have only been shown to be valid through these tests up to these smaller compute budgets. Furthermore, the non-optimal test models were exclusively four times larger and smaller than the optimal models. This testing standard is less stringent than lower factors would be. A more comprehensive validation of the scaling laws would have included a larger number of non-optimal models closer in size to the optimal models. Appendix B explores this for EEG data with a few additional non-optimal test models. Despite the testing compute budget limitations, the scaling laws should continue to be useful for researchers scaling BIOT [43] up past the compute budgets used to test them as there are currently no other scaling laws available to guide such work. They should also be useful as a starting point for other researchers who want to apply further fine-tuning and optimization to the scaling laws to improve their accuracy.

Compute resources also limited the number of training runs which could be done when

developing the scaling laws. For comparison, Hoffman *et al.* [16] were able to train a sufficient variety of models to get iso-FLOP curves with clear minima over the range of 3 orders of magnitude. In this thesis, approximately 1 order of magnitude was covered for each dataset instead. This could have been a contributing factor to the difference between the two scaling laws developed for the EEG dataset.

5.2 Future Work

Important research which would be complimentary to what is documented in this thesis is the study of dataset size scaling requirements as is done for language models by Kaplan *et al.* [20]. Some tests were done in Section 3.4 to determine if the size of the datasets used in this thesis were likely to limit performance. Tests showed that that was likely not the case. However, overfitting should eventually occur at some higher compute budget with a large enough model and a sufficient number of steps. Therefore, for further scaling up of BIOT [43], it will be important to develop a scaling law to predict dataset size requirements based on compute budget.

Another area for future work is to study the robustness of the developed scaling laws at larger compute budgets. This could lead to improved scaling laws or confirm that the scaling laws presented in Table 5.1 are valid past the compute budgets used for testing them in this thesis. Furthermore, Hoffman *et al.* [16] conjecture based on their results that at very high compute budgets, the true scaling law exhibits some curvature which pulls it away from being a strict power law. It is unclear whether this would be the case for BIOT as well or at what compute budget that may occur. Future studies could explore this.

Additionally, a scaling law which predicts the resulting loss decrease from a given compute budget increase, as is developed by Kaplan *et al.* [20] for language models, would also be useful for researchers in combination with laws which predict fine-tuning performance for certain tasks based on pre-training loss. This would allow decisions about compute budget to be made with resulting fine-tuning performance in mind. For example, if a particular application required a known level of performance, the compute budget could be chosen such that the performance target is met without expending resources on additional compute needlessly.

Unfortunately, a generally applicable law to predict fine-tuning performances based on pre-training loss would be very difficult to develop. In this thesis, it is assumed that lower pre-training loss indicates better utility from pre-training. However, this will only be the case when the pre-training dataset and task are relevant to the fine-tuning task in question.

Even when this is the case, specific relationships between pre-training loss and fine-tuning performance are likely to vary across different fine-tuning tasks. For example, it has been found to be the case for large language models to suddenly exhibit new emergent abilities with few-shot prompting after different sufficiently large scales are reached [40]. Thus, it is likely that depending on the difficulty of the task, that different minimum levels of pre-training loss on different pre-training datasets would be required before improvements from pre-training are even observed. It may also be the case for BIOT that previously unsolved biosignal problems could be solved effectively with a large enough increase in scale.

Some work is done in Appendix C which fine-tunes some BIOT models pre-trained with the TUH-EEG dataset [28] on both the TUAB [34] and TUEV [2] datasets. The goal of those tests is to provide some sample of the effects of both varying compute budgets and varying numbers of parameters/training tokens within constant compute budgets on fine-tuning performance. But, with only one pre-training dataset, 2 fine-tuning datasets, and a limited maximum compute budget, generally applicable rules about such trade-offs are not proposed and are left for future work, as described above.

Finally, the scaling laws developed in this thesis can be used in work to develop large foundational models for biosignal processing tasks as Yang *et al.* [43] call for in the conclusion to their work on BIOT.

References

- [1] National health and nutrition examination survey data. Centers for Disease Control and Prevention (CDC). National Center for Health Statistics (NCHS) Hyattsville, MD: U.S. Department of Health and Human Services, Centers for Disease Control and Prevention, 2014. https://wwwn.cdc.gov/Nchs/Nhanes/2013-2014/PAXHR_H.htm#Component_Description.
- [2] Harati A, Golmohammadi M, Lopex S, Obeid I, and Picone J. Improved eeg event classification using differential energy. *IEEE Signal Process Med Biol Symp*, 2015.
- [3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [4] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [5] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

- [6] Erick A Perez Alday, Annie Gu, Amit J Shah, Chad Robichaux, An-Kwok Ian Wong, Chengyu Liu, Feifei Liu, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, et al. Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020. *Physiological measurement*, 41(12):124003, 2020.
- [7] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, Jorge Luis Reyes-Ortiz, et al. A public domain dataset for human activity recognition using smartphones. In *Esann*, volume 3, page 3, 2013.
- [8] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020.
- [9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.
- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018. cite arxiv:1810.04805Comment: 13 pages.
- [11] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12124–12134, 2022.
- [12] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [13] Fuchang Gao and Lixing Han. Implementing the nelder-mead simplex algorithm with adaptive parameters. *Computational Optimization and Applications*, 51(1):259–277, 2012.
- [14] Wendong Ge, Jin Jing, Sungtae An, Aline Herlopian, Marcus Ng, Aaron F Struck, Brian Appavu, Emily L Johnson, Gamaleldin Osman, Hiba A Haider, et al. Deep ac-

tive learning for interictal ictal injury continuum eeg patterns. *Journal of neuroscience methods*, 351:108966, 2021.

- [15] Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer. *arXiv preprint arXiv:2104.01778*, 2021.
- [16] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. Training compute-optimal large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30016–30030. Curran Associates, Inc., 2022.
- [17] Edward Hu and Greg Yang. Maximal update parametrization and hyperparameter transfer, 2022. [Source code]. <https://github.com/microsoft/mup>.
- [18] Jin Jing, Wendong Ge, Shenda Hong, Marta Bento Fernandes, Zhen Lin, Chaoqi Yang, Sungtae An, Aaron F Struck, Aline Herlopian, Ioannis Karakis, et al. Development of expert-level classification of seizures and rhythmic and periodic patterns during eeg interpretation. *Neurology*, 100(17):e1750–e1762, 2023.
- [19] Dinesh John, Qu Tang, Fahd Albinali, and Stephen Intille. An open-source monitor-independent movement summary for accelerometer data processing. *Journal for the Measurement of Physical Behaviour*, 2(4):268–281, 2019.
- [20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- [21] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.
- [22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [23] Yoav Levine, Noam Wies, Or Sharir, Hofit Bata, and Amnon Shashua. Limits to depth efficiencies of self-attention. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan,

- and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22640–22651. Curran Associates, Inc., 2020.
- [24] Hongli Li, Man Ding, Ronghua Zhang, and Chunbo Xiu. Motor imagery eeg classification algorithm based on cnn-lstm feature fusion network. *Biomedical signal processing and control*, 72:103342, 2022.
 - [25] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
 - [26] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. Large language models: A survey. *arXiv preprint arXiv:2402.06196*, 2024.
 - [27] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis: proceedings of the biennial Conference held at Dundee, June 28–July 1, 1977*, pages 105–116. Springer, 2006.
 - [28] Iyad Obeid and Joseph Picone. The temple university hospital eeg data corpus. *Frontiers in Neuroscience, Section Neural Technology*, 10, 196, 2016.
 - [29] Wei Yan Peh, Yuanyuan Yao, and Justin Dauwels. Transformer convolutional neural networks for automated artifact detection in scalp eeg. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 3599–3602. IEEE, 2022.
 - [30] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.
 - [31] Ali Hossam Shoeb. *Application of machine learning to epileptic seizure onset detection and treatment*. PhD thesis, Massachusetts Institute of Technology, 2009.
 - [32] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.
 - [33] Yonghao Song, Xueyu Jia, Lie Yang, and Longhan Xie. Transformer-based spatial-temporal feature learning for eeg decoding. *arXiv preprint arXiv:2106.11170*, 2021.

- [34] Lopez de Diego Sylvia. Automated interpretation of abnormal adult electroencephalograms. *Temple University Libraries*, 2017.
- [35] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.
- [36] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
- [37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2023.
- [38] Prateek Verma and Jonathan Berger. Audio transformers: Transformer architectures for large scale audio understanding. adieu convolutions. *arXiv preprint arXiv:2105.00335*, 2021.
- [39] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I Lunze, Wojciech Samek, and Tobias Schaeffter. Ptb-xl, a large publicly available electrocardiography dataset. *Scientific data*, 7(1):1–15, 2020.
- [40] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.
- [41] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [42] C Yang, D Xiao, MB Westover, and J Sun. Self-supervised eeg representation learning for automatic sleep staging. arxiv 2021. *arXiv preprint arXiv:2110.15278*.
- [43] Chaoqi Yang, M Westover, and Jimeng Sun. Biot: Biosignal transformer for cross-data learning in the wild. *Advances in Neural Information Processing Systems*, 36, 2024.

- [44] Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.
- [45] Guo-Qiang Zhang, Licong Cui, Remo Mueller, Shiqiang Tao, Matthew Kim, Michael Rueschman, Sara Mariani, Daniel Mobley, and Susan Redline. The national sleep research resource: towards a sleep data commons. *Journal of the American Medical Informatics Association*, 25(10):1351–1358, 2018.

APPENDICES

Appendix A

TUH-EEG Channels

The channel names that were kept from the [TUH-EEG \[28\]](#) dataset are shown in Table [A.1](#).

Table A.1: The 24 channels that were selected to be kept from the [TUH-EEG](#) [28] dataset. Names are taken directly from the dataset.

Channel Name
EEG FP1-REF
EEG FP2-REF
EEG F4-REF
EEG C3-REF
EEG C4-REF
EEG P3-REF
EEG P4-REF
EEG O1-REF
EEG O2-REF
EEG F7-REF
EEG F8-REF
EEG T3-REF
EEG T4-REF
EEG T5-REF
EEG T6-REF
EEG F3-REF
EEG CZ-REF
EEG FZ-REF
EEG PZ-REF
EEG T2-REF
EEG T1-REF
EEG EKG1-REF
EEG A1-REF
EEG A2-REF

Appendix B

Additional EEG Analysis

A plot is shown here which compares the test results for the scaling laws developed in Sections 4.4 and 4.5 for the TUH-EEG [28] dataset. In addition to the results of these tests, some additional models were trained up to the same compute budget of 500 billion for additional context. This effectively results in a very sparse iso-FLOP curve for the test compute budget. The plot can be seen in Figure B.1.

There is not a sufficiently clear minimum in order to make conclusions about the comparative accuracy of either scaling law and it is outside of the scope of this thesis to do so. But, this plot is included here, in the appendix to provide another visual aid to represent the results of the scaling law tests for the EEG data and to show the relative performance of other model sizes for the compute budget used for testing the EEG scaling laws.

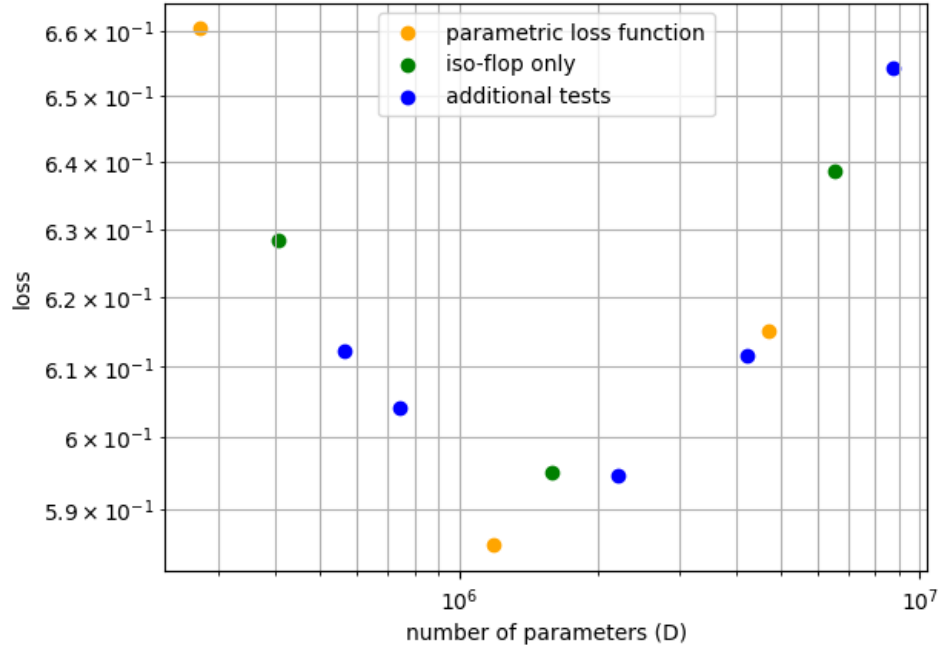


Figure B.1: Combined results of each [EEG](#) scaling law test from Tables [4.8](#) and [4.10](#). Results from additional tests, not documented in any table, are also plotted. All points are the final loss of models trained with a compute budget of $C = 500 \times 10^9$.

Appendix C

Fine Tuning Analysis

In this chapter, the effects of pre-training with the [TUH-EEG](#) dataset on fine-tuning tasks are studied. The [TUAB](#) [34] and [TUEV](#) [2] datasets, used previously by Yang *et al.* [43] when evaluating [BIOT](#), are defined and used for fine-tuning. For each dataset, two studies are conducted. One such study varies the compute budget used for pre-training from 500 billion (as in Sections 4.4 and 4.5) to 100 billion, to 20 billion. The scaling law developed in Section 4.5 is used to choose the balance of number of parameters and number of training tokens to maintain optimality. The second study compares different balances of number of parameters and number of training tokens for a constant compute budget of 500 billion. The pre-trained models from Section 4.5.3 are used. Finally, for each pre-trained model which is fine-tuned, randomly initialized models are also fine-tuned in order to study the effects of pre-training on the fine-tuning performance for each dataset.

Results show that increasing compute budget of pre-training does improve performance on both fine-tuning datasets. Some discussion is provided on the generalization of this result to other datasets and the potential presence of a performance ceiling for pre-training compute budgets approaching infinity. Results for a constant compute budget with varying model sizes and number of training steps used in pre-training show that a model which is larger than optimal may perform better than the predicted optimal model size. This result is discussed in detail with reference to Kaplan *et al.*'s [20] work on modeling loss based on model size, dataset size, and compute budget for large language models. It is concluded that the scaling laws developed in this thesis should still be useful for the development of large foundation models for biosignal transformers.

C.1 Datasets

C.1.1 Temple University Hospital Abnormal

The **TUAB** dataset [34] consists of 2,717 train **EEG** recordings, 1,346 of which are labeled abnormal and 1,371 of which are labeled normal. It also includes 276 evaluation recordings, 150 of which are labeled as normal while the other 126 are labeled as abnormal. After applying the same preprocessing as was applied to the **TUH-EEG** dataset [28] (retaining 24 channels from Appendix A and resampling to 256Hz), 295,596 train samples, 76,542 validation samples, and 36,945 test samples were generated. All samples are 10 seconds long each.

The dataset is a subset from the **TUH-EEG** dataset [28] which was labeled as normal or abnormal based on available reports that accompanied the recordings.

C.1.2 Temple University Hospital Events

The Temple University Hospital Events [2] dataset is also a subset of the **TUH-EEG** [28] dataset. It consists of 370 **EEG** recordings all from different patients. Events were manually labeled by neurologists into 6 types. These classes are: spike and/or sharp waves (SPWS), periodic lateralized epileptiform discharges (PLED), generalized periodic epileptiform discharges (GPED), artifacts (signals which are not cerebral in origin such as spikes from removing/replacing electrodes), eye-movement, and background (other).

The dataset is provided already split into a train set with 290 recordings and a test set with the remaining 80 recordings. The train set follows approximately this distribution: 0.8% SPSW, 7.4% GPED, 13.4% PLED, 1.4% eye movement, 13.2% artifact, and 63.9% background [2]. The test set follows a similar distribution.

The pre-processing script provided by Yang *et al.* [43] was modified to maintain the channels documented in Appendix A and used to extract 83,932 5 second training samples and 26,067 5 second test samples. All the recordings are by default resampled to 250Hz. These were thus resampled to 256Hz to be compatible with the pre-trained models from this work.

C.2 Training and Testing Procedure

Both the training procedures for the [TUAB](#) dataset and [TUEV](#) datasets are quite similar. Commonalities between both procedures will be discussed first, followed by attributes which are specific to each.

Yang *et al.* [43] fine-tune on both the [TUAB](#) [34] and [TUEV](#) [2] dataset and provide their training scripts for those tests. Thus, the training procedures used here is very similar to theirs. However, some adaptations were made to work with the pre-trained models targeted by the scaling laws developed herein. For instance, maximal update parametrization was used for the pre-trained models which means they use $1/d_{model}$ attention scaling rather than $1/\sqrt{d_{model}}$ attention scaling. Thus, the pre-trained models which were fine-tuned maintained that scaling. For consistency, the randomly initialized models also used $1/d_{model}$ attention scaling. The other aspects of maximal update parametrization are maintained for all tests as well such as using the mup implementation of AdamW [17] and adjusting weight initializations as described in Section 3.2.

Another difference from the training procedure used by Yang *et al.* [43] is that early stopping based on validation performance is not used. It was found that allowing the fine-tuning process to use either of the entire datasets for training resulted in any performance improvements from pre-training being eclipsed by the fine-tuning itself. That is to say that performance did not improve when using a pre-trained model vs a randomly initialized model when pre-training with the whole dataset and with early stopping enabled. Therefore, in order to better study the effects of pre-training on the available datasets and fine-tuning tasks, tests were conducted multiple times with different prescribed numbers of steps. Lower number of steps allow observations with improvements from pre-training to be recorded. For each study, all tests are repeated with 25, 100, 400, and 1600 steps.

This approach was chosen in favor of reducing the dataset size by some factor and continuing to use early stopping because it is unclear if early stopping with a validation set would be an optimal regularization strategy with a smaller dataset because the validation set size would be reduced as well. With a smaller validation set size, the signal for stopping will become less representative of the model’s true generalized performance, resulting in stopping occurring more and more randomly. Furthermore, it was chosen that one epoch with a smaller dataset should be completed rather than repeating data for multiple epochs to avoid concerns about overfitting in the absence of the early stopping regularization method.

Finally, learning rate and weight decay are left at the same values used by Yang *et al.* [43] in their experiments of 0.001 and 10^{-5} respectively.

Using these training parameters, each configuration is trained 20-40 times with different random seeds. The random seed affects the training data loader shuffling of all models and the random initialization of models which are not pre-trained. The mean performance score for each configuration is then recorded in following sections.

C.2.1 TUAB

For the TUAB dataset specifically, a batch size of 256 instead of the 512 used by Yang *et al.* [43] in order for training to be done on available devices. With this batch size, 1 epoch of training on the training dataset consists of 1154 steps.

The loss function used for optimization is binary cross-entropy between the predicted and true labels (abnormal or normal).

C.2.2 TUEV

The batch size used for the TUEV dataset was 128 which is the same as what Yang *et al.* [43] used in their experiments. 599 steps are required for one pass through the training set with this batch size.

The loss function used for optimization is simply cross entropy loss between the predicted class probabilities and the true classes.

C.3 Constant Compute Budget—TUAB

This section compares the fine-tuning performance of models which were pre-trained using the same pre-training compute budget. The fine-tuning task is binary classification of 10 second EEG recordings from the TUAB dataset into either normal or abnormal. The performance is reported with the mean AUROC. A higher score indicates better performance.

Three different pre-trained models are used. These are the smaller, optimal and larger models used to test the EEG scaling law developed using the parametric loss function method in Section 4.5.3. Versions of each of these models which are not pre-trained (randomly initialized) are also trained in order to study the overall effect of pre-training on those model sizes. Tables C.1, C.2, C.3, and C.4 display the results of these tests.

Of particular note for this study, is that improvements from pre-training are only consistently observed in Tables C.1 and C.2 where 25 and 100 steps of fine-tuning are completed

Table C.1: Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 25 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.7741 ± 0.0136	0.8021 ± 0.0117
Optimal ($d_{model} = 152$)	0.7891 ± 0.0112	0.8217 ± 0.0072
Larger ($d_{model} = 304$)	0.7943 ± 0.0094	0.8245 ± 0.0067

Table C.2: Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 100 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.8156 ± 0.0074	0.8262 ± 0.0069
Optimal ($d_{model} = 152$)	0.8176 ± 0.0053	0.8354 ± 0.0069
Larger ($d_{model} = 304$)	0.8202 ± 0.0039	0.8361 ± 0.0085

respectively. In both cases, the smaller model is decidedly outperformed by both the larger and optimal pre-trained models. However, the larger model slightly outperforms the optimal model in each case.

C.4 Varying Compute Budget–TUAB

This section compares the fine-tuning performance of models which were pre-trained using compute budgets of 500 billion, 100 billion, and 20 billion. The dataset and task are the same as the previous section and performance is again reported by average AUROC.

The scaling law for EEG developed using the parametric loss function method was used to select optimal balances of numbers of parameters and training steps for each compute budget. For 20 billion, a model with $d_{model} = 72$ and 273,336 parameters was chosen. The

Table C.3: Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 400 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.8366 ± 0.0055	0.8202 ± 0.0091
Optimal ($d_{model} = 152$)	0.8374 ± 0.0066	0.8241 ± 0.0118
Larger ($d_{model} = 304$)	0.8380 ± 0.0066	0.8447 ± 0.0089

Table C.4: Comparison of mean AUROC across models with a constant compute budget fine-tuned on the TUAB dataset with 1600 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.8226 ± 0.0102	0.7840 ± 0.0118
Optimal ($d_{model} = 152$)	0.8239 ± 0.0086	0.7958 ± 0.0111
Larger ($d_{model} = 304$)	0.8234 ± 0.0094	0.7976 ± 0.0099

Table C.5: Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 25 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.7741 ± 0.0136	0.8109 ± 0.0094
100 billion	104	0.7857 ± 0.0108	0.8191 ± 0.0071
500 billion	152	0.7891 ± 0.0112	0.8217 ± 0.0072

corresponding number of steps was 73,170. For a compute budget of 100 billion, a model with $d_{model} = 104$ and 561,208 parameters was selected. This models was pre-trained for 178,187 steps. For the compute budget of 500 billion, the optimal test model from Section 4.5.3 was used ($d_{model} = 172, D = 1,185,016, S = 421,935$). Versions of each of these models which are not pre-trained are also tested in order to study the effects of pre-training with different numbers of fine-tuning steps. The results of these tests for 25, 100, 400, and 1600 fine-tuning steps are displayed in Tables C.5, C.6, C.7, and C.8 respectively.

Of note in this study is again that improvements from pre-training are only consistently observed for 25 and 100 steps. Although the model with the smallest compute budget does improve with pre-training in the tests done with 400 steps. In the tests done where pre-training is beneficial, the models which are pre-trained with larger compute budgets outperform those with smaller pre-training compute budgets, which is expected.

Table C.6: Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 100 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.8156 ± 0.0074	0.8292 ± 0.0069
100 billion	104	0.8168 ± 0.0057	0.8351 ± 0.0081
500 billion	152	0.8176 ± 0.0053	0.8354 ± 0.0069

Table C.7: Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 400 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.8366 ± 0.0055	0.8400 ± 0.0083
100 billion	104	0.8358 ± 0.0086	0.8301 ± 0.0134
500 billion	152	0.8374 ± 0.0066	0.8241 ± 0.0118

Table C.8: Comparison of mean AUROC across models with varying compute budgets fine-tuned on the TUAB dataset with 1600 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.8226 ± 0.0102	0.8039 ± 0.0120
100 billion	104	0.8205 ± 0.0120	0.7963 ± 0.0108
500 billion	152	0.8239 ± 0.0086	0.7958 ± 0.0111

C.5 Constant Compute Budget–TUEV

This section compares the fine-tuning performance of models which were pre-trained using the same pre-training compute budget. The fine-tuning task is multi-class classification of 5 second EEG recordings from the TUEV dataset into one of the 6 classes described in Section C.2.2. The performance is reported with Cohen’s Kappa score. A higher score indicates better performance.

The set of tests done in this section mirror those in Section C.3 other than the use of the TUEV dataset. The results are displayed in Tables C.9, C.10, C.11, and C.12.

Of particular note for this study, is that the smaller pre-trained model never exhibits any performance improvements over its randomly initialized counterparts in any of the four tests with varying numbers of fine-tuning steps. Improvements from pre-training for the optimal model is only observed when fine-tuning with 25 and 100 steps. In both of these tests, the optimal model is slightly outperformed by the larger model. The larger model improves with pre-training for all numbers of steps. It can also be noted that in these same tests, the performance of the randomly initialized models clearly improves with higher sizes.

Table C.9: Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the [TUEV](#) dataset with 25 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.3485 ± 0.0506	0.3208 ± 0.0683
Optimal ($d_{model} = 152$)	0.3575 ± 0.0323	0.4015 ± 0.0339
Larger ($d_{model} = 304$)	0.3786 ± 0.0350	0.4127 ± 0.0447

Table C.10: Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the [TUEV](#) dataset with 100 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.3904 ± 0.0372	0.3531 ± 0.0432
Optimal ($d_{model} = 152$)	0.3720 ± 0.0313	0.4055 ± 0.0334
Larger ($d_{model} = 304$)	0.3909 ± 0.0311	0.4169 ± 0.0341

Table C.11: Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the [TUEV](#) dataset with 400 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.3962 ± 0.0279	0.3414 ± 0.0419
Optimal ($d_{model} = 152$)	0.3822 ± 0.0280	0.3885 ± 0.0381
Larger ($d_{model} = 304$)	0.3801 ± 0.0291	0.4119 ± 0.0264

Table C.12: Comparison of mean Cohen’s Kappa score across models with a constant compute budget fine-tuned on the [TUEV](#) dataset with 1600 steps.

Model description	Randomly Initialized	Pre-trained from Section 4.5.3
Smaller ($d_{model} = 72$)	0.3908 ± 0.0247	0.3077 ± 0.0407
Optimal ($d_{model} = 152$)	0.3810 ± 0.0307	0.3810 ± 0.0385
Larger ($d_{model} = 304$)	0.3699 ± 0.0269	0.3959 ± 0.0301

Table C.13: Comparison of mean Cohen’s Kappa score across models with varying compute budgets fine-tuned on the [TUEV](#) dataset with 25 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.3485 ± 0.0506	0.3689 ± 0.0266
100 billion	104	0.3632 ± 0.0382	0.3891 ± 0.0456
500 billion	152	0.3575 ± 0.0323	0.4015 ± 0.0339

Table C.14: Comparison of mean Cohen’s Kappa score across models with varying compute budgets fine-tuned on the [TUEV](#) dataset with 100 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.3904 ± 0.0372	0.3743 ± 0.0338
100 billion	104	0.3933 ± 0.0341	0.3941 ± 0.0396
500 billion	152	0.3720 ± 0.0313	0.4055 ± 0.0334

C.6 Varying Compute Budget–[TUEV](#)

This section compares the fine-tuning performance of models which were pre-trained using compute budgets of 500 billion, 100 billion, and 20 billion. The dataset and task as the same as the previous section and performance is again reported by average Cohen’s Kappa score.

The same pre-trained models from Section C.4 were used in this section as well. Results for 25, 100, 400, and 1600 fine-tuning steps are displayed in Tables C.13, C.14, C.15, and C.16 respectively.

Of note in this section is that the pre-trained model with the smallest compute budget does show an improved performance for the 25 step test but not for the others. The only test which shows consistent improvement across all models is the test which uses 25 steps. In this same test, the models with larger pre-training compute budgets outperform others.

Table C.15: Comparison of mean Cohen’s Kappa score across models with varying compute budgets fine-tuned on the [TUEV](#) dataset with 400 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.3962 ± 0.0279	0.3724 ± 0.0231
100 billion	104	0.3859 ± 0.0358	0.4327 ± 0.0339
500 billion	152	0.3822 ± 0.0280	0.3885 ± 0.0381

Table C.16: Comparison of mean Cohen’s Kappa score across models with varying compute budgets fine-tuned on the [TUEV](#) dataset with 1600 steps.

Compute Budget	d_{model}	Randomly Initialized	Pre-trained
20 billion	72	0.3908 ± 0.0247	0.3732 ± 0.0242
100 billion	104	0.3701 ± 0.0357	0.3872 ± 0.0449
500 billion	152	0.3810 ± 0.0307	0.3810 ± 0.0381

C.7 Analysis and Discussion

A common theme in all sets of tests is that pre-training is not usually observed to improve performance when completing 400 or more steps of fine-tuning. This suggests that the fine-tuning datasets are not small enough for pre-training on the [TUH-EEG](#) dataset [28] with the compute budgets and pre-processing configurations used in this work to be useful in improving performance. In order to study the trade-offs of varying compute budgets and varying balances of number of parameters and pre-training tokens for a constant compute budget, the tests in which pre-training improved performance should be observed.

Regarding the trade-off of fine-tuning performance for different computes budgets, it is clear in Tables [C.5](#), [C.6](#), [C.13](#), [C.14](#) that the models pre-trained with larger compute budgets outperform those with smaller compute budgets. The degree to which performance increases seems to be larger between compute budgets of 20 billion and 100 billion than between 100 billion and 500 billion, indicating that a saturation level may eventually be reached. Of course, the highest point at which this saturation level could be is 1 for both [AUROC](#) and Cohen’s Kappa since those are the maximum scores for each metric.

However, other factors may lead to this saturation level being even lower than 1. For instance, the ground truth labels in the fine-tuning dataset may not have an error rate of 0 since they both rely on human annotations. This would make a perfect score nearly impossible for any model trained and tested on a held-out set from the dataset. Another factor which may impose a ceiling on the fine-tuning performance of the models given a pre-training compute budget approaching infinity, is the size of the pre-training dataset. In this work, the pre-training dataset size is not infinite. In fact, for the compute budgets used, some models were trained on as many as 40 epochs of data. Although no overfitting was detected in Section 3.4 for [EEG](#) models, with a large enough model and sufficient data repetition, the benefits of pre-training should saturate and lead to a lower max fine-tuning performance in the absence of new, diverse pre-training data.

Given infinite and relevant pre-training data and an infinite compute budget, it is unclear if the maximum performance would be equivalent to the performance of human

annotators. The final factor which could potentially limit the performance is the size of the fine-tuning dataset itself. Having access to an infinitely large fine-tuning dataset would likely render pre-training unnecessary as pre-training is typically used as a supplement to a task that has limited labeled data for training. But that does not mean that any arbitrarily small amount of pre-training data is enough to achieve max performance. With 295,596 train samples for TUAB [34] and 83,932 train samples for TUEV [2] it seems unlikely that the fine-tuning dataset size for either task would be a limiting factor for max performance given infinite limits of compute budget and pre-training dataset size. However, given the imbalance in the TUEV dataset, with only 0.8% of recordings being labeled SPSW [2], it is a possibility.

The tests which compare different balances of number of parameters and number of training tokens show improvements from pre-training with 100 or 25 steps of fine-tuning in Tables C.1, C.2, C.9, C.10. However, note that the smaller pre-trained model ($d_{model} = 72$) does not show improvement for the TUEV [2] task in any test. In fact, the pre-training is detrimental in this case. This is interesting because the mirrored tests for TUAB [34] did show improvement with the exact same pre-trained model for 25 and 100 steps along with the other model sizes. This may indicate that the smaller model did experience some level of overfitting on the pre-training dataset in which it learned a small set of specific features which were useful for the TUAB [34] dataset but not for TUEV [2] dataset. In fact, the model used for the smallest compute budget in the varying compute budget tests also had $d_{model} = 72$ but was trained for fewer steps and ended up with an improved performance over randomly initialized for TUEV [2] in Table C.13. Perhaps this model had not yet reached the point of specializing into specific features and had still learned only general features useful for most fine-tuning tasks.

Regardless of the detrimental effect of pre-training with the smaller model for TUEV, the pre-trained models for both TUAB [34] and TUEV [2] have similar relative performances. Specifically, the optimal and larger models outperform the smaller models by a large margin. This is as expected given the pre-training loss of each model—the highest of which was that of the smaller model. However, the larger models outperform the optimal models by a slight margin despite having a higher pre-training loss in Section 4.5.3. Upon applying a two-tailed homoscedastic t-test to these results, the difference between the means is not shown to be statistically significant. The results of these t-tests can be seen in Table C.17. Note that a p-value of 0.05 or less is typically considered to show statistical significance of the difference between the means of two sets of samples.

These results suggest that the trade-offs related to the balance of number of parameters, and number of training tokens for a fixed compute budget are not as simple as those related to maintaining optimal configurations with varying compute budgets. Specifically, model

Table C.17: T-tests comparing mean of optimal pre-trained models vs larger pre-trained models.

Dataset	Steps	P-value	Number of samples
TUAB	25	0.073	40
TUAB	100	0.718	40
TUEV	25	0.401	20
TUEV	100	0.292	20

size seems to have a role in influencing fine-tuning performance independently from the pre-training compute budget. This can be shown by observing the performances of the randomly initialized models shown in Tables C.1, C.2, C.9, C.10 where the fine-tuning performance is clearly different for different sizes of models (improving with size except for Table C.10). When taking into consideration the different performances of the randomly initialized models, the optimally pre-trained models improve over their randomly initialized counterparts in all 4 of Tables C.1, C.2, C.9, C.10 by larger margins than both the larger and smaller models do. So, optimizing a scaling law to minimize pre-training loss seems to have successfully resulted in models which benefit most from pre-training, but which may not always significantly outperform other models which were pre-trained sub-optimally.

Despite this result, I still recommend using the scaling laws developed in this thesis to inform compute budget allocations for the development of foundation models for biosignals. This is because, as pre-training compute budgets increase and multiple larger pre-training datasets are used, the benefits of pre-training optimally should become more important than the benefits of fine-tuning with a larger model. Furthermore, with fine-tuning datasets not increasing in size, the improvement in performance coming strictly from the fine-tuning process will be limited by the dataset size itself, and not by the model size which will be much larger for a foundation model with a large compute budget. This prediction is based on the work of Kaplan *et al.* [20] who found that dataset size, model size, and compute budget were the 3 main factors which determined model performance for large language models. They also claim that when one of these parameters is held constant while the others are allowed to increase significantly, then the one parameter limits performance independently. In the case of a large foundation model for biosignals, the fine-tuning dataset size will eventually become the most limiting factor for the benefits gained from fine-tuning alone, which will reduce the effect of model size on final performance. In other words, if the models being compared are all large enough that they reach near maximum performance from random initialization on the fine-tuning dataset, then when they are pre-trained before fine-tuning, the pre-training loss should be the most important factor

in determining final improved performance rather than model size.