Pretraining Large Language Models for Finnish

University of Turku
Department of Computing
Master of Science Thesis
TurkuNLP
May 2025
Risto Luukkonen

UNIVERSITY OF TURKU
Department of Computing

Risto Luukkonen: Pretraining Large Language Models for Finnish

Master of Science Thesis, 63 p., 4 app. p.
TurkuNLP
May 2025

Transformers have revolutionized the field of natural language processing (NLP) and remain as the dominant large language model (LLM) architecture. However, openly available models have had limited support for low-resourced languages, with most of the work focusing on high-resource languages such as English, where pretraining datasets consists of hundreds of billions or even trillions of words. This work investigates the challenges of training a language model specifically for Finnish, a language spoken by only less than 0.1% of the world population. The training dataset is compiled from a combination of web crawls, news, social media and eBooks. This work explores two distinct approaches to pretraining: (1) training a family of monolingual models from scratch (186M to 13B parameters) named FinGPT and (2) continual pretraining of the multilingual BLOOM model on a mix of its original training data accompanied with Finnish, resulting in a 176 billion parameter model we call BLUUMI.
To evaluate model performance, this work introduces FIN-bench, a Finnish adaptation of BIG-bench — a widely used benchmark for language model evaluation. Models are then evaluated using FIN-bench, along with additional assessments for toxicity and bias.

Keywords: Transformer, Large Language Model, Natural Language Processing, Distributed computing, Megatron-LM, DeepSpeed

Contents

1 Introduction **1**

2 Background **4**

2.1 Machine Learning 4

2.1.1 Optimization 7

2.1.2 Normalization 9

2.1.3 Regularization 10

2.1.4 Transfer learning 11

2.2 NLP 12

2.2.1 Tokenization 14

2.2.2 Word Embeddings 15

2.3 Transformer 16

2.3.1 Attention 18

2.3.2 Embeddings and softmax 21

2.3.3 Positional encoding 22

2.3.4 Position-wise Feed Forward Networks 23

2.3.5 Encoder-only models 24

2.3.6 Encoder-Decoder models 25

2.3.7 Decoder-only models 26

2.3.8 Scalability 28

2.4 Distributed training . . . 30
2.4.1 Mixed precision training . . . 31
2.4.2 Data parallelism . . . 31
2.4.3 Tensor parallelism . . . 32
2.4.4 Pipeline parallelism . . . 34
2.4.5 Zero Redundancy Optimizer . . . 35

3 Data **39**
3.1 Data sources . . . 39
3.2 Preprocessing . . . 42
3.3 Tokenization . . . 43
3.4 Sampling . . . 44

4 Pretraining **45**
4.1 Models . . . 45
4.2 Requirements . . . 46
4.3 Software . . . 47
4.4 Hardware . . . 48
4.5 Debugging . . . 49

5 Evaluation **51**
5.1 FIN-bench dataset . . . 52
5.2 Few-shot results . . . 54
5.3 Alignment . . . 57
5.4 Bias . . . 58
5.5 Toxicity . . . 59

6 Discussion **60**

7 Conclusions **62**

8 Further work **63**

References **64**

Appendices

A **A-1**

A.1 Timespan covered by Finnish datasets A-1

A.2 Comparison of mC4-Fi and CC-Fi datasets A-1

A.3 Toxicity scores A-3

A.4 Data distribution by source before and after weighting A-3

A.5 FIN-bench examples A-4

List of Figures

2.1 Artificial neuron with n inputs and one output 6

2.2 Structure of a single training step: 1) forward pass, 2) performance evaluation, 3) backward pass, 4) weight update. (Chollet, 2021) . . . 7

2.3 Transformer model architecture (Vaswani et al., 2017) 17

2.4 Attention weights for the word `it` 18

2.5 Flash Attention tiling mechanism (left) and performance improvement (right) versus the original attention (Dao et al., 2022) 21

2.6 T5 training objective, where consecutive spans of words are masked with sentinel tokens <**X**> and<**Y**> (Raffel et al., 2020) 25

2.7 BERT-style full self-attention (left) compared to the GPT-style causal self-attention (Alammar, 2018b) 27

2.8 GPT-style triangular attention masking (Alammar, 2018b) 27

2.9 GPT-3 scaling power-law (Brown et al., 2020) 28

2.10 *Chinchilla scaling laws* for compute-optimal scaling demonstrated the overparametrization of the previous models (Kaplan et al., 2020) . . . 29

2.11 Chinchilla-optimal token-parameter scaling (Hoffmann et al., 2022) . 30

2.12 Mixed precision training dynamics (Micikevicius et al., 2018) 31

2.13 Megatron-LM implementation of tensor parallelism. f acts as an identity operator in the forward pass and as a *all reduce* in the backward pass. g works as a *all reduce* in the forward pass, as an identity operator in the backward pass (Shoeybi et al., 2019) 33
2.14 GPipe-schedule (Narayanan et al., 2021) 35
2.15 PipeDream-Flush-schedule (Narayanan et al., 2021) 35
2.16 Memory reduction on different ZeRO-stages versus the baseline (Rajbhandari et al., 2019) 38

4.1 Exploding loss curves in the beginning 49
4.2 Impact of different features and codebases during debugging. 50

5.1 Validation losses with 5-point moving average smoothing. 51
5.2 Overall FIN-bench evaluation results 55
5.3 BLOOM and BLUUMI performance on FIN-bench with random baseline (dotted line). 56
5.4 176B model performance on English evaluations. 56
5.5 HHH-alignment of all models with random baseline (dotted line). . . 57
5.6 Gender bias of 13B model predictions on occupation holder vs statistics from the Statistics Finland. 58
5.7 Unprompted toxicity of Finnish models. 59

List of Tables

3.1 Data sources. 40

3.2 Preprocessed data statistics, weights, and ratios by source. 44

4.1 Model architectures 46

4.2 Pretraining hyperparameters. 46

A.1 Examples of Fin-BENCH tasks A-4

List of acronyms

ANN Artificial Neural Network

FNN Feedforward Neural Network

LLM Large Language Model

MLP Multilayer Perceptron

NLP Natural Language Processing

GEMM General Matrix Multiplication

SOTA State-of-the-art

1 Introduction

Written text is a commonly accepted system of certain peculiar symbols positioned closely next to each other. Based on the positions of these symbols, in relation to others in the sequence, a lump of symbols forms a meaning, a word. When multiple words occur together, in relation to others in the sequence, they form a context. This context is essential to specify the exact meaning of each word as some words can have varying meaning depending on the context. These kinds of traits underline the very nature of natural language. It can be syntactically very variable, varying between dialects and languages, but allow conveying the same semantic meaning by syntactically very different text.

Processing written natural language has historically been a very difficult problem for computers. However, since natural language is a central medium for storing and passing information, it is necessary to have the appropriate technology available to process it. Furthermore, the amount of digital text available has been rapidly increasing since the dawn of the World Wide Web, and the more data there is, the more essential it is to have advanced tooling to process it.

Natural Language Processing (NLP) is an interdisciplinary field of computer science and linguistics focusing on technology and methodology that enables computers to process and interact with natural language. It includes tasks such as counting word frequencies, classifying text into predefined categories, or generating open-ended answers to questions. Modern techniques use massive quantities of data to

train advanced language models that learn to generalize contextualized understanding from billions of words in a self-supervised manner, allowing the model to write human-like text and perform tasks such as summarizing articles based on the users' requests. Most of the open models have focused on performance in high-resourced languages such as English, and language coverage for low-resource languages is very limited.

The topic of this thesis is pre-training large language models (LLMs) in Finnish using one of Europe's most powerful supercomputers LUMI[1]. This thesis attempts to answer two research questions. The first is more technical but simple in its nature: *What techniques can be used to train a large language model on a super computer?* Another one considers the data available for training, as Finnish can be considered a small language, and asks *Can we train an LLM for Finnish when we have so little data?* Additionally, since during the training we had access to the full capacity of LUMI for a brief time, we try to explore the continuous pre-training setup to train Finnish to an existing model by training a massively multilingual behemoth, 176B parameter BLOOM, and ask *Is it possible to leverage an already pre-trained model to teach it Finnish?*

Chapter 2 introduces the relevant background including general theory behind machine learning and NLP, followed by more technical chapter about the currently dominant Transformer-architecture. It also elaborates on the distributed training methods that were used to train the models introduced in this work. Chapter 3 discusses the data used for the model training; sources used, pre-processing and the challenges we faced concerning the amount of data available for Finnish. Chapter 4 discusses the actual technical work done for this thesis. We go through the choice of model architectures and software framework and hardware used for training. We follow up with explaining the pre-training process, then proceed to Chapter 5 and

[1]https://eurohpc-ju.europa.eu/supercomputers/our-supercomputers_en

go through our evaluation setup and results. In Chapter 6 we discuss our results, reflecting the decisions made in Chapter 4, following up with conclusions in Chapter 7, and finishing up by discussing future work in Chapter 8 that has been motivated by the results of this work.

The work introduced in this thesis (data work, training and evaluating) was done between the fall of 2022 and the early 2023 and published in an article *FinGPT: Large Generative Models for a Small Language* (Luukkonen et al., 2023), where I was the leading author. Chapters 3, 4, and 5 closely follow the content and structure of the corresponding sections in that article. The selected methods were based on the latest scientific results on the topic at the time. The field has seen substantial advancements since and newer methods, such as supervised fine-tuning and Reinforcement Learning with Human Feedback (RLHF), are left out of the scope of this work. As multiple more advanced models have been introduced since, the specific models, datasets, and evaluations presented in this work should be regarded as somewhat outdated, and treated more as a snapshot of the field at that time.

I have used generative AI during the proofreading phase of this work, once the thesis was otherwise complete. Its contribution was to point out grammatical errors and help smoothen the flow of my text, not to produce actual content. All the suggestions, when found useful, were applied manually and with consideration.

2 Background

2.1 Machine Learning

> "A computer program is said to learn from **experience E** with respect to some class of **tasks T** and **performance measure P** if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997)

This is a well-known, high level definition for machine learning from the book *Machine Learning* by Mitchell (1997). For example, if the task **T** is to classify images into cats and dogs, we can measure the performance **P** as the proportion of the correct predictions made by the algorithm. By showing more images and making step-by-step adjustments using a learning rule, then by definition the model learns if its prediction accuracy increases over time.

There is a wide range of different machine learning methods varying from very simplistic and transparent to increasingly more complex, opaque, and computationally intensive techniques. Simple methods, such as k-nearest neighbor classifiers (Cover & Hart, 1967), store the training data and make the prediction based on the similarity between a new sample and the stored examples. The training data consists of the features we want to use for classification, here referred to as *dimensions*. Memory requirements for this kind of classifier can be very lightweight, requiring at minimum only $N \times D$ of memory, where N is the number of training samples

and D is the number of dimensions in each sample. To perform inference on such a model, the naive algorithm needs to simply go through each of the data points, compute its similarity to the current input using a distance metric, and find the top K most similar ones. The label assigned to the current data point would be the most frequent class of these most similar *nearest neighbors*. These models are easy to implement, lightweight, and can run on modest hardware.

This is very different for modern deep neural networks. These models can contain billions, even trillions of parameters, far exceeding the GPU memory of a single device. The largest models require terabytes of GPU memory for training. These models require dedicated hardware and engineering for both training and inference, along with vast quantities of training data.

This work focuses on neural network architecture called *transformers*, which is currently dominant in the field of large language models. Other machine learning methods are narrowed outside of the scope of this work.

Artificial Neural Networks (ANN) get their name from being analogous to human neurons; each computational unit receives a number of input signals, typically from units in the preceding layer of similar units, and produces an output. A perceptron (Equation 2.1) is a specific type of ANN that computes a linear combination of a real-valued input vector and outputs 1 when the result is above some threshold, -1 otherwise. Figure 2.1 illustrates a perceptron, assuming the activation function f is defined to implement the binary threshold logic. A single-layer perceptron can be used to model primitive Boolean functions that are linearly separable (AND, OR, NAND, NOR, NOT). Mathematically, it can be expressed as the following:

$$o(x_1, ..., x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + w_n \cdot x_n > 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.1}$$

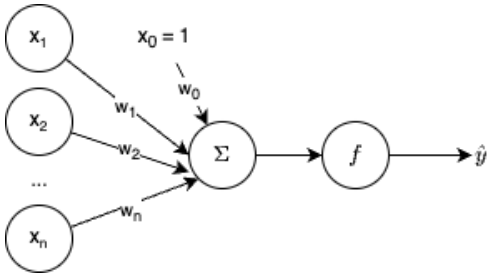


Figure 2.1: Artificial neuron with n inputs and one output

The values w_1, ... w_n are called *weights* and are the parameters optimized by the learning algorithm. The value w_0 represents a learnable bias weight for the bias unit x_0.

These Boolean functions can be represented by a single decision boundary. In contrast, functions such as XOR are not linearly separable and cannot be modeled by a single-layer perceptron. To increase representational power beyond linearly separable functions and to handle increasingly complex problems, multiple units can be densely interconnected to form layers. This multi-layer generalization is known as multilayer perceptron (MLP). It is a fully connected type of Feedforward Network (FNN), a type of non-cyclic ANN consisting of multiple layers in which computational units propagate values forward through successive layers to the neurons in the output layers.

To increase the representational power and make proper use of these multiple layers, various different activation functions can be used in place of the binary threshold, depending on the use case. A typical activation function for binary classifier is the *sigmoid* function, while the *softmax* function is commonly used for the multi-class use case. (Mitchell, 1997; Pajankar & Joshi, 2022)

2.1.1 Optimization

A training step for a neural network requires adjustment of model weights with respect to some training data. This involves computing gradients for each weight via *backpropagation* and then applying a learning rule.

A training step consists of a forward pass, performance evaluation, a backward pass and parameter update, as illustrated in Figure 2.2. During the forward pass, input data is propagated through all the layers in the neural network. Each layer receives either the raw input data or the transformed outputs (*activations*) from the previous layer. It then applies a transformation using its activation function, weights, and biases. As the data flow reaches the final layer, its output is interpreted as the model's prediction. This predicted value is used to assess the model's performance by comparing it against the ground truth using a *loss function*. The choice of loss function depends on the type of task. Common examples include mean squared error (MSE) for regression tasks and cross-entropy for classification.

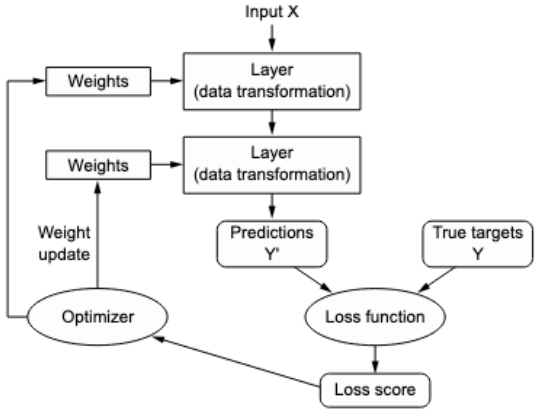


Figure 2.2: Structure of a single training step: 1) forward pass, 2) performance evaluation, 3) backward pass, 4) weight update. (Chollet, 2021)

Backpropagation utilizes the chain rule of calculus to compute how each variable in the network affects the final output. It starts by computing the gradient of the loss function with respect to the activations of the output layer. Since these activations depend on the previous layer, the gradients must also be propagated

backward through the network using the chain rule — all the way to the first layer. The primary purpose of computing gradients is to get the direction and magnitude of the steepest slope for a function. To improve the performance of the network, the learning algorithm needs to take a step into the opposite direction, as its aim is to minimize, not maximize, the loss function. When using naive gradient descent, the weight (and bias) update rule is simple: the gradients are multiplied by some learning rate η and the product is then simply subtracted from the current weights w_i as in the formula $w_i \leftarrow w_i - \eta \cdot \frac{\partial L}{\partial w_i}$.

The learning rate dictates the magnitude of the optimization step. It can be either static or change as a function over time. When training a large neural network with a large, high-quality dataset, using a static learning rate with the naive gradient descent leads to slow convergence, since the optimizer lacks the ability to dynamically adapt the step size based on the seen data during training.

Multiple more sophisticated algorithms have been developed to improve the convergence speed. AdaGrad (Duchi et al., 2011) demonstrates the idea of element-wise adaptive learning rate by keeping a running sum of squared gradients and dividing the learning rate by that sum. As the sum grows and the training progresses, this leads to smaller and smaller updates, which also can become a problem on the longer runs. RMSProp (Hinton et al., 2012) is similar, but uses an exponentially decaying moving average of the squared gradients, helping it to adapt to recent gradients. The moving average makes the learning rate more stable and balanced, and the decaying factor makes it dependent on only a certain number of past samples, making it more suitable for longer runs.

Adam (Kingma & Ba, 2015) is currently one of the most popular optimizers. It borrows ideas from RMSProp and AdaGrad and introduces two additional variables, called the first and second moments. The weight update rules are shown in Eq. 2.2, where θ represents the weights, g_t denotes the gradients, and β_1 and β_2 are the

exponential decay rates for the moment estimates. The first moment $\hat{m}_t$ calculates the moving average of the gradients g_t, while the second moment $\hat{v}_t$ computes the moving average of their squared values g_t^2. The first moment helps smooth out potentially noisy updates from the past, while the second the second moment adjusts the learning rate based on the magnitude of the gradients.

$$\begin{aligned} g_t &= \nabla_\theta f(\theta_t) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, \quad v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \end{aligned} \tag{2.2}$$

2.1.2 Normalization

When training a neural network, it is important to maintain a stable flow of data through the layers. If no control mechanism for input data or hidden states is applied, subsequent layers may end up amplifying the activations and trigger butterfly effects. Eventually, outlier values can easily cause the optimization process to fail, with gradients approaching zero or exploding to extremely large values. A smooth flow of gradients also boosts model convergence.

Numerical stability can be addressed at multiple points in the training pipeline. First, one can try to find good initial weights. Proper initialization has been shown to result in more stable gradients, such as the variance based initialization proposed by He et al. (2015). Second, input data can be preprocessed to be on a suitable scale, for example by standardizing it, which can either be incorporated in the model as an integral part or used as a separate preprocessing step. Third, activations within the model can be normalized. The choice of normalization method depends on the model architecture, but two common methods are batch normalization and layer normalization, which both are additional stabilization layers of the model.

Batch normalization standardizes activations using the mean and variance of the *mini-batch*–a small subset of data used in a single training step. It has proven to be effective, but has the drawback of being dependent on batch size and sequence length (Ioffe & Szegedy, 2015). **Layer normalization** is invariant to batch size or sequence lengths, producing consistent results during both training and testing. Activations are normalized over the hidden dimension, resulting in values with a mean near to 0 and variance close to 1, as in batch normalization (Ba, 2016). Layer normalization is computationally expensive and has been rivaled by the more efficient **Root Mean Square Layer Normalization** (RMSNorm), which achieves comparable performance but offers a runtime reduction by 7%–64% compared to Layer Normalization, depending on the model (Zhang & Sennrich, 2019). Normalization layers are an integral part of most modern architectures.

2.1.3 Regularization

Regularization refers to techniques used to reduce the risk of overfitting a model to its training data. An overfitted model may perform well when evaluated on the training data, but underperform on the test data, even if they are both sampled from the same distribution. In such cases, the model fails to generalize to new samples as it has only memorized the training data, making it ineffective.

Roughly, there are three prominent approaches to regularization: reducing the model capacity, using a combination of multiple models or increasing the amount of training data.

As model size often correlates with performance, keeping the model size small is a one way to limit its capacity. A common method in neural networks is to use dedicated dropout layers. These layers deactivate parts of the network during training, forcing the model to generalize over multiple neural paths. These layers are deactivated during evaluation and inference (Hinton, 2012; N. Srivastava et

al., 2014). As modern LLMs are not considered to be data-constrained, dropout has become largely unnecessary, since datataset sizes are counted in billions or in trillions of tokens and model parameters in billions. However, it has been suggested to be beneficial during the initial-phase of training and result in models with lower loss (Z. Liu et al., 2023). Another common requlariza tion technique is to add a small penalty to the loss function. Popular choices are L1 and L2-requlariza tion (Hoerl & Kennard, 1970; Tibshirani, 1996), the latter being frequently used with the Adam-optimizer as a component known as *weight decay*.

2.1.4 Transfer learning

Large neural networks have one interesting and particularly useful property: they can be trained on one domain and that knowledge can be adapted to another, related domain. This means that a single strong baseline model can be adapted to multiple tasks. Since training large neural networks demands substantial computational resources, having to do the heavy pre-training only once is advantageous, as it can be performed by an organization with vast computational resources. Fine-tuning a model for specific tasks is often a much more light-weight process, requiring only a fraction of the data and compute of the pre-training. Transfer learning is not limited to any specific modalities and has been shown to work well with both visual and language models.

For example, AlexNet, a convolutional neural network architecture introduced by Krizhevsky et al. (2012), demonstrated state-of-the-art performance in image classification. The authors first trained it on 15 million images with 22k categories to acquire a general level image classifier. Then they fine-tuned it on 1.4 million task-specific images from the ILSVRC-2012 (Russakovsky et al., 2015) challenge dataset, outperforming other competing models.

Word2vec (Mikolov, Corrado, et al., 2013) demonstrated that semantic informa-

tion can be encoded into word embedding vectors and that these features can be utilized in other tasks such as semantic clustering. BERT (Devlin et al., 2019) was pre-trained on a large corpus of unlabeled web-crawled text, and later fine-tuned for a specific task with task-specific data and a suitable classification head. It achieved state-of-the-art performance in the GLUE benchmark (Wang et al., 2018), which was the leading NLP benchmark at the time. Furthermore, T5 (Raffel et al., 2020) demonstrated that a single pre-trained model can be fine-tuned for any combination of tasks as long as they are formulated as text-to-text problems.

Transfer learning and task generalization are especially notable in generative transformer models (Section 2.3.7) which have shown strong performance in zero- and few-shot scenarios. State-of-the-art pre-trained models can often perform well on an unseen task by simply including a few examples of the task in the input prompt (Brown et al., 2020; Radford et al., 2019).

2.2 NLP

Natural Language Processing (NLP) has long been a central and quickly evolving field within computer science. It includes many important and essential applications, such as automatic text classification, information extraction, machine translation, and question answering.

As natural language exhibits myriad variations, it is a difficult domain to process with programs that follow predefined, handwritten rules. Early NLP methods were purely symbolic, able to handle tasks such as machine translation but only with a limited set of variations. To address that, some tasks aim to reduce syntactic variation to make e.g. information retrieval from a corpus easier. For example, stemming and lemmatization aim to reduce inflected word forms to their base forms, enabling a more straightforward approach to statistical analysis.

Modern machine learning methods have become considerably more robust to

variations in data, thanks to the billions or trillions of tokens processed during pre-training and the development of powerful neural models. Transformers have excelled at previously challenging classification tasks such as named entity recognition (NER) and relation extraction. Named entity recognition is the task to classify words into predefined classes for a given input text. For instance, consider the classes 'organization' and 'person': a classifier can be trained to label words in a sequence, aiming to tag the correct instances of these classes. Relation extraction refers to classifying a pair of tagged entities within a sequence by assigning them a relation from a predefined set, such as 'founder-of'. A pre-trained BERT-model (Devlin et al., 2019) can be fine-tuned on annotated datasets to achieve high performance on these tasks. For example, FinBERT demonstrated a performance increase in F1-score from 87% to 92% outperforming the previous non-transformer models on NER (Virtanen et al., 2019).

Encoder-only models (e.g., BERT), designed for understanding and classification tasks, excel at span-based question answering. In contrast, decoder-only models (e.g., GPT), which focus on text generation, are better suited for open-ended tasks such as question answering, where the model generates a sequence of tokens instead of selecting a span from the input. Although this can lead to potential hallucinations, it also provides flexibility and more natural expression, allowing multiple disjoint pieces of text to be combined into a single coherent sentence. Another relevant open-ended task is summarization, where the ability to rephrase key pieces of information is important. As decoder-style generative models have become increasingly powerful, they now excel in more demanding open-ended tasks, such as code generation and mathematical reasoning.

2.2.1 Tokenization

Tokenization is a data pre-processing method where input text is broken down into words or sub-words and converted into a list of token IDs using a look-up table. This look-up table, known as the *vocabulary*, is constructed in a separate process using a set of rule-based pre-processing methods. These pre-tokenization steps typically include splitting by whitespace and regular expressions that handle punctuation and other special characters.

Several algorithms exist for constructing the lookup table, such as WordPiece (Song et al., 2021), Unigram (Kudo, 2018) and Byte-pair Encoding (BPE) (Gage, 1994; Sennrich et al., 2016). The tokenizer used in this work uses byte-level BPE-algorithm, as in GPT-2 and GPT-3 (Brown et al., 2020; Radford et al., 2019). This approach operates directly on UTF-8 encoded strings, allowing it to handle any text without encountering out-of-vocabulary conditions. Unlike other algorithms that map unrecognized tokens to a shared token ID, byte-level BPE ensures that all text inputs can be represented by encoding them as byte sequences. This work adopts the byte-level BPE algorithm and the following section details the principles and mechanics of the BPE algorithm.

BPE-algorithm

The underlying logic of the byte-level BPE algorithm is analogous to character-based tokenization, and for clarity, this explanation focuses on the character-level variant. Assume we have a corpus for tokenizer creation. First, the corpus is pre-processed to identify word boundaries, typically by splitting text based on whitespace and punctuation. This step outputs a list of words. The initial vocabulary is created by collecting all unique characters in the corpus. We then compute the frequencies of adjacent character pairs (e.g., ('a', 'b')). The most frequent pair is selected, merged into a new token (e.g., 'ab'), and added to both the vocabulary and the list of merge

rules. This merge rule is applied to the corpus, resulting in a slightly compressed corpus, and the token pair frequencies are updated accordingly. At this point, one new token and one merge rule have been created. To create the complete tokenizer, we need to iterate this process until a desired stopping condition is met, such as achieving a fixed vocabulary size or reaching a predefined number of merges.

Once the vocabulary and merge rules are finalized, the tokenizer can be applied to new text. The input is first split into the smallest base units – in this case, individual characters. Merge rules, stored as an ordered list, are applied greedily: they are iterated over until the first matching rule is found and applied. This process continues until no further merges are possible. The resulting tokens are then mapped to their corresponding indices by using the vocabulary as a look-up table, resulting in a list of integers.

2.2.2 Word Embeddings

Word embeddings are an essential method in LLMs to encode the contextualized meaning of words numerically, capturing both syntax and semantics.

They can be defined as a mapping of a word from a vocabulary V to a real-valued vector in an embedding space of dimensionality D (Schnabel et al., 2015). These are fixed-sized representational vectors for each token in the vocabulary that encode distributional features of words. The idea behind embeddings is based on the distributional hypothesis: words that occur in similar contexts tend to share similar meanings (Almeida & Xexéo, 2019). For example, words *dog* and *cat* or *king* and *queen* have clear semantic relationships. Properly trained embeddings for these words are expected to have similar vector representations. Semantically similar words tend to locate closely in the embedding space, which can be used in many NLP tasks such as clustering and parsing, or as standalone features for machine learning models.

Word embeddings are an integral part of modern NLP models such as transformers. They usually form the first layer of a neural network, mapping input values from token sequences to continuous representations, followed by an arbitrary number of hidden states, and in token prediction models, mapped back to vocabulary probabilities via an output embedding layer.

2.3 Transformer

Transformers were introduced in 2017 by the paper *Attention Is All You Need* in Vaswani et al. (2017), originally designed for machine translation. The architecture was rapidly adopted, revolutionizing the NLP field by achieving state-of-the-art (SOTA) performance on a multitude of tasks such as machine translation, question answering and text classification. Today, transformers are the dominant architecture in NLP.

The biggest breakthrough came from the massively parallelizable architectural design, which reduced relative compute costs to a fraction of what the pre-transformer SOTA models required (Vaswani et al., 2017). Unlike earlier sequential models like LSTMs and RNNs, which processed input tokens one at a time in order, transformers removed sequential bottleneck by injecting positional information directly into token embeddings. This enabled the model to process each token in the sequence simultaneously with the attention mechanism (Section 2.3.1) handling the contextual relatioships.

The original Transformer consists of two function stacks, shown in Figure 2.3: an encoder and a decoder. The encoder maps a symbolic input sequence $\mathbf{X}$ into a continuous latent representation $\mathbf{Z}$. Using cross-attention over $\mathbf{Z}$, the decoder generates the output sequence $Y = (y_1, ...y_n)$ one element at a time, by auto-regressively appending the newly generated element y_{n+1} to the output, which is then fed back to the model as input when generating the next element. The encoder consists of

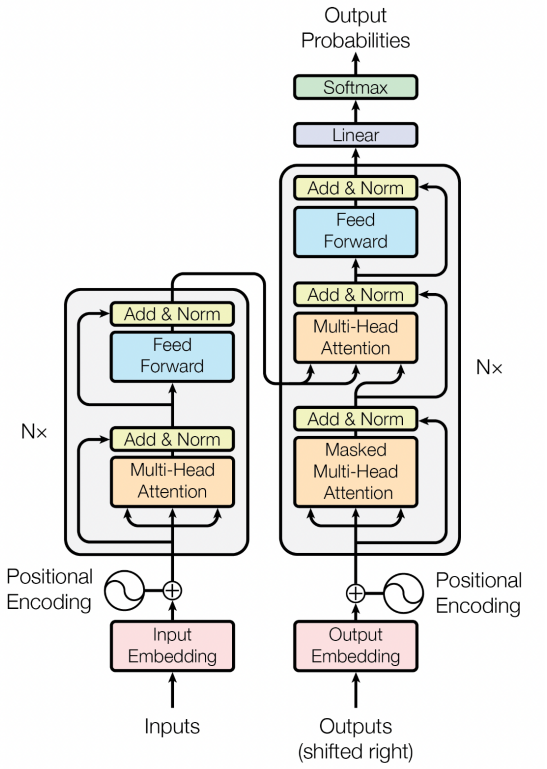


Figure 2.3: Transformer model architecture (Vaswani et al., 2017)

multiple identical layers each containing two main sublayers: multi-headed attention and a position-wise fully connected feedforward network (FFN). Both sublayers employ residual connections followed by layer normalization.

The decoder stack is similar, but it introduces an additional intermediate sublayer that applies multi-head attention over the encoder's output. The self-attention layer is masked to allow only attention over the previous embeddings, preventing the model from attending the future tokens while training.

This flexible architecture can be adapted for different model types: encoder-only models (e.g., BERT), decoder-only models (e.g., GPT), or full encoder-decoder models (e.g., T5).

2.3.1 Attention

The original transformer architecture uses a function called *scaled dot-product attention*:

$$\text{Attention(Q, K, V)} = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Self-Attention is the key building block that makes the transformer powerful. It can be thought of as a function that calculates the relevance of the other words in a sequence to a given target word, as illustrated in Figure 2.4. For example, in the *"The animal didn't cross the street because it was too tired"* the word `it` refers to `animal`. Self-attention allows the model to learn this contextual information directly from the training data. The attention function operates on three matrices: keys (K), queries (Q) and values (V). The queries and keys are multiplied, and the result is divided by the scaling factor $\sqrt{d_k}$ to ensure that softmax gradients stay within a favorable numerical range. The resulting weights are then multiplied by the values V to produce the final output. Because these operations are predominantly general matrix

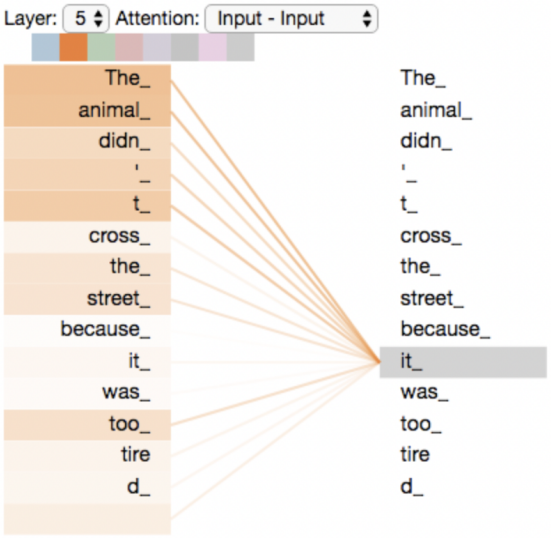


Figure 2.4: Attention weights for the word `it`
(Alammar, 2018a)

multiplications (GEMMs), they can efficiently leverage GPU-optimized libraries for

parallel computation. To take further advantage of parallelism and enhance the model's representational power, attention is enhanced to *multi-head attention* (Eq 2.3), where h represents the number of heads.

$$\begin{aligned} \text{MultiHead(Q,K,V)} &= \text{Concat}(head_1, ..., head_h)W^h \\ \text{where } head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^W) \end{aligned} \tag{2.3}$$

Here, the queries, keys, and values are projected into h separate subspaces through learned projection matrices W_i^Q, W_i^K, W_i^W, each performing their own attention computations independently. The outputs are then concatenated and passed through the final linear projection. Multi-head attention allows model to learn and jointly attend different representational subspaces at different positions, which would be impossible with only one head due to averaging (Vaswani et al., 2017).

Attention variations

Self-attention requires calculating the attention score of all tokens in a sequence for each individual token in a sequence. Therefore, the function is quadratic in time and memory complexity with respect to the sequence length. This introduces computational and memory challenges when processing long sequences. Various methods have been proposed to enable a long context length.

Approximation The first line of research has focused on reducing the number of required computation operations by introducing a variety of approximation methods. Most of these approaches focus on sparsifying the attention matrix. Sparse Transformers (Child et al., 2019) introduced an attention factorization that operates across multiple time-steps, reducing computational complexity to $O(n\sqrt{n})$, while maintaining or even surpassing the performance of the original transformer. Similar sparse attention blocks were used in GPT-3 (Brown et al., 2020).

The Longformer (Beltagy et al., 2020) succesfully experimented with a combination of global attention with local sliding window attention patterns to achieve linear scaling with respect to input sequence length. It was trained on sequences up to 23k tokens and performed well on long-document tasks.

CoLT5 (Ainslie et al., 2023) applied selective logic which "light" and "heavy" blocks. Light local attention is applied to all tokens, while heavy full attention was used only for conditionally selected tokens, improving computation efficiency. This logic is also applied to feed-forward layers. The authors reason that not all the tokens are equally important and thus those of more importance should be given more compute. They report a less-than-linear scaling, enabling fast and efficient processing of contexts up to 64k tokens.

Optimization Another line of work has focused on optimizing the attention function. Rabe and Staats (2021) observed that by applying the distributive property, single-query attention can be reformulated as shown in Eq 2.4, where s_i represents the score for each key.

$$s_i = \text{dot}(q, k_i),\ s'_i = e^{s^i},\ \text{attention}(q, k, v) = \frac{\sum_i v_i s'_i}{\sum_j s'_j} \tag{2.4}$$

When expanded to full self-attention by applying it sequentially for all the queries, the original $O(n^2)$ space complexity is reduced to $O(\log n)$. This reduction occurs because the full attention matrix is not required to be materialized in memory. The time complexity remains the same as all the operations of full self-attention are still the same but the matrix is computed in a different order to reduce memory usage.

Flash-attention (Dao, 2023; Dao et al., 2022) is a fused GPU kernel that offers both reductions in memory usage and improvements in throughput. Its tiling mechanism prevents the materialization of the full intermediate $n\ x\ n$ attention

matrix by looping through blocks of **K** and **V** matrices and loading them into fast SRAM (static random access memory). As SRAM is much faster than DRAM (dynamic random access memory), FlashAttention not only saves memory, but also significantly speeds up computations as shown in Figure 2.5.[1]

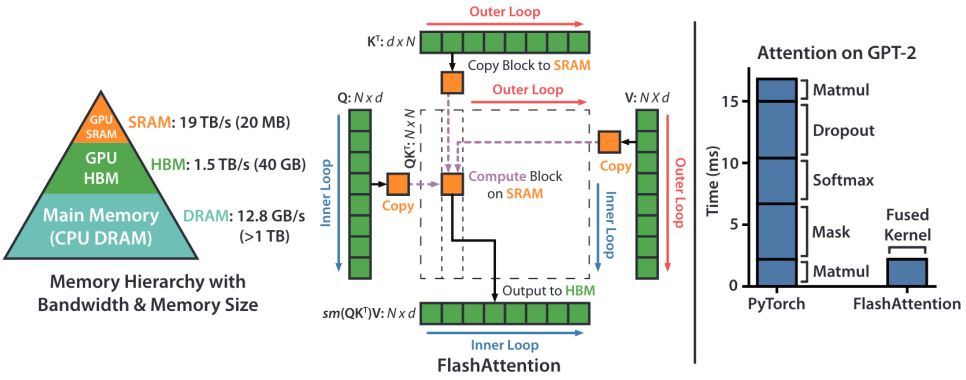


Figure 2.5: Flash Attention tiling mechanism (left) and performance improvement (right) versus the original attention (Dao et al., 2022)

2.3.2 Embeddings and softmax

Transformers use learned embeddings, meaning that embedding parameters are also learned during model training. The embedding layer is a $v \times d$ dimensional layer which maps the input sequence of tokenized text into $s \times d$ dimensional representations, where $\boldsymbol{d}$ is the hidden layer size, $\boldsymbol{v}$ is the vocabulary size and $\mathbf{s}$ is the sequence length. The output activations from the embedding layer are then propagated through the model, layer-by-layer, up to the output layer. In GPT-like decoder-only models, trained on a next-token prediction task, the final output layer projects the hidden dimensions back to the vocabulary size, producing logits for the next token. Vaswani et al. (2017) introduced weight tying of the output projection layer with those of the input embedding layer to reduce the model's parameter count

[1]FlashAttention support for AMD hardware was released after the models discussed here were trained, and was not used in this work.

and improve generalization by sharing weights between the input and output layers. However, many modern frontier models prefer using separate, untied embeddings.

2.3.3 Positional encoding

Since self-attention in transformers is order-independent, positional information must be explicitly added to the input sequence. The original transformer paper by Vaswani et al. (2017) experimented with two methods for adding positional information: *absolute* and *learned positional embeddings*, and chose the former since the authors did not find any meaningful difference between the two. The absolute positional embedding method encodes positional information using sine and cosine functions with frequencies as shown in Equation 2.5, where *pos* refers to the token position and i refers to the dimension:

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad \mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \quad (2.5)$$

This positional information is then added to the outputs of the embedding layer and passed through the model via residual connections. However, absolute positional encoding has two main weaknesses: 1) it produces the same encoding for any token at position i, leading to different embeddings when positions are shifted, for example, when the same sentence is prefixed with a common phrase like "in fact". 2) It suffers from weak length extrapolation (Press et al., 2021): when the model has been trained with sequences of 512 tokens, its performance degrades dramatically when inferencing on longer sequences as the model lacks the ability to infer with new, unseen positional signals.

Several alternative methods have addressed these issues. Roformer (Su et al., 2021) introduced the Rotary Positional Encoding (RoPE), which improves conver-

gence by injecting positional signals into the hidden states at every layer, rather than only at the embedding layer. On the other hand, T5 (Raffel et al., 2020) uses Relational Positive Bias, which is shown to have some extrapolation capabilities (Press et al., 2021). Attention values for query and key-matrices get a shared, learned bias, which is applied at each layer. This bias is dependent on the distance between the query and key tokens. As this method adds new parameters to each layer, it is also significantly slower to train. ALiBi (Attention with Linear Biases) (Press et al., 2021) avoids the overhead of additional parameters by using a static, non-learned bias. This bias is added to the outputs of the query-key dot-product: $softmax(\mathbf{q}_i\mathbf{K}^T + m \cdot [-(i-1), ..., -2, -1, 0])$, where scalar m is a head-specific slope that is fixed before training. The authors argue that this method has stronger extrapolation capabilities than the others, but Dey et al. (2023) report performance degradation on sequence lengths surpassing 12% of the training sequence length.

2.3.4 Position-wise Feed Forward Networks

Each layer in both the encoder and decoder contains a fully connected feed-forward network (FFN) following the attention mechanism. The FFN applies two linear transformations with a ReLU activation in between (Vaswani et al., 2017).

This feed-forward network is defined formally in Equation 2.6:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \tag{2.6}$$

where W_1, W_2, b_1, and b_2 are learned parameters. The feed-forward network operates independently and identically at each position, introducing non-linearity and enabling richer feature transformations.

2.3.5 Encoder-only models

Models built using only transformer encoder stacks gained widespread adoption following the release of **BERT** (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) (Devlin et al., 2019). Pre-trained encoder-only models can be fine-tuned for specific tasks by simply adding an output layer on top. The original BERT was a monolingual English model, and it was quickly followed by monolingual models in other languages, including Finnish (Virtanen et al., 2019), as well as multilingual models such as mBERT[2].

In contrast to GPT-like (Radford et al., 2019) decoder-only models that utilize only past tokens for context, BERT-like models are able to attend to both left and right contexts simultaneously (Devlin et al., 2019). The authors argue that unidirectional models are suboptimal for sentence- and token-level tasks. BERT pre-training was performed by randomly masking 15% of the input tokens with a special `MASK`-token, forcing the model to predict the true token behind the mask. This is also known as masked language modeling. As an auxiliary task, next-sentence prediction was also introduced, but it was later shown to have limited impact (Y. Liu et al., 2019).

BERT-like models are popular in NLP systems, commonly used for tasks such as semantic search (Nayak, 2019), and word- and sequence-level text classification (Min et al., 2023). Domain-specific variants have also been developed, including SciBERT for scientific literature (Beltagy et al., 2019), BioBERT for biomedical texts (Lee et al., 2020), MusicBERT for music modeling (Zeng et al., 2021), and MedBERT (Rasmy et al., 2021) for medical records.

[2]https://github.com/google-research/bert/blob/master/multilingual.md

2.3.6 Encoder-Decoder models

The original transformer with an encoder-decoder architecture aimed at building a model suitable for machine translation (Vaswani et al., 2017). Later research has shown that this architecture is highly capable in transfer learning settings across many tasks, resulting in models such as T5 (Raffel et al., 2020) from the paper "*Exploring the Limits of Transfer Learning with Transformers*". The authors demonstrated that, rather than using a separate task-specific model head for each task, multiple tasks could be reformulated as text-to-text problems. Thus, a single model with a unified loss function and decoding procedure could handle a wide variety of text tasks while achieving performance comparable to task-specific architectures.

The pre-training objective for T5 is masked-span prediction. Short consecutive spans of text are randomly masked and replaced with sentinel tokens as shown as `<X>` and `<Y>` in Figure 2.6. These sentinel tokens are special placeholders used during training and do not correspond to any real tokens, as the objective is to predict only the masked sequences "hidden" behind them.

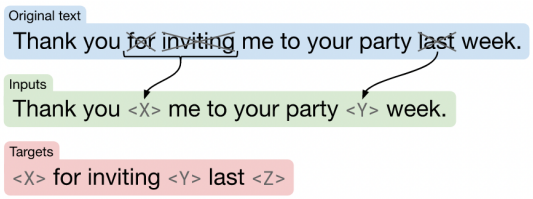


Figure 2.6: T5 training objective, where consecutive spans of words are masked with sentinel tokens <**X**> and<**Y**> (Raffel et al., 2020)

Subsequent research has expanded on the encoder-decoder setup, combining common masking and denoising strategies from T5 (Raffel et al., 2020), BERT (Devlin et al., 2019) and GPT (Radford et al., 2019) into a unified pre-training pipeline (Tay et al., 2022) resulting in performance gains, and dramatically improving the generalization capabilities for unseen tasks with instruction fine-tuning (Chung et al., 2024). Since the T5 architecture includes a decoder-component, it can be used

as a generative, causal language model. This raises a question about the most feasible architecture for generative tasks. Tay et al. (2022) discuss this and conclude that it is unclear which model architecture could be considered universally superior. However, their results suggest that encoder-decoder models should be preferred over decoder-only models if – and only if – there is no concern about parameter counts. They also emphasize that the pre-training objective may be, in fact, intrinsically more important than the specific model architecture.

2.3.7 Decoder-only models

The decoder-only models consist of the right-hand side of the original transformer architecture shown in Figure 2.3. This architecture was first introduced with the Generative Pretrained Transformer (GPT) (Radford et al., 2018), and later expanded upon with GPT-2 (Radford et al., 2019) and GPT-3 (Brown et al., 2020). These models are defined by their next-token prediction objective.

Suitable for a wide range of text-to-text tasks, the model takes a sequence of tokens as input and predicts the next plausible token. Sequences are generated in a loop, where the newly generated token is appended to the input, and the model predicts the next token in the updated sequence. This process is repeated until a stopping criterion is met, such as predicting a special "end-of-sequence" token. This method is called auto-regression, and models using it are often referred to as auto-regressive language models.

The prediction head uses a softmax function to output probabilities over the entire vocabulary. During inference, these outputs can be decoded using different sampling methods such as greedy decoding, where the token with the highest probability is selected, or alternative strategies that introduce variation by sampling from the probability distribution. The training objective is formulated as a multi-class prediction task using one-hot encoded labels, with logits passed through a cross-

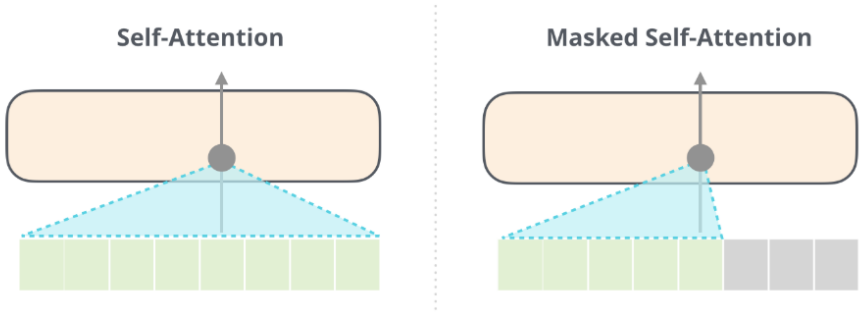


Figure 2.7: BERT-style full self-attention (left) compared to the GPT-style causal self-attention (Alammar, 2018b)

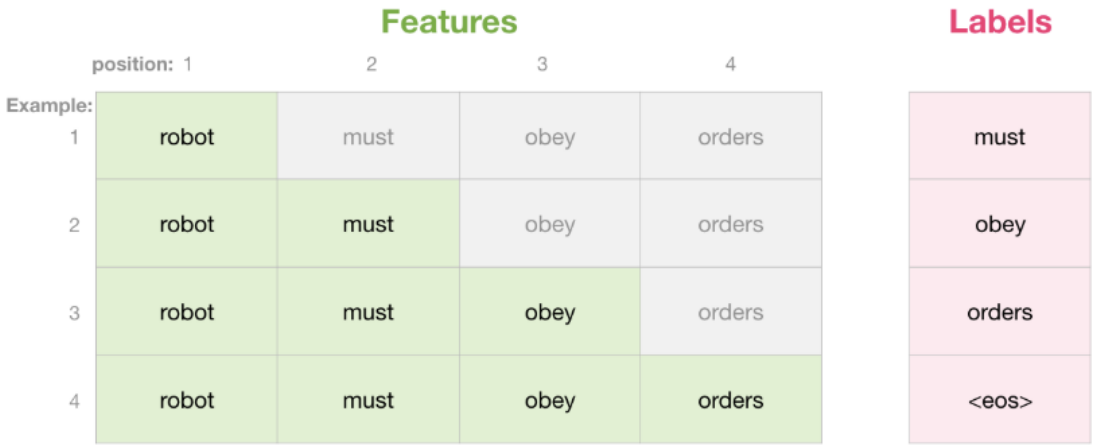


Figure 2.8: GPT-style triangular attention masking (Alammar, 2018b)

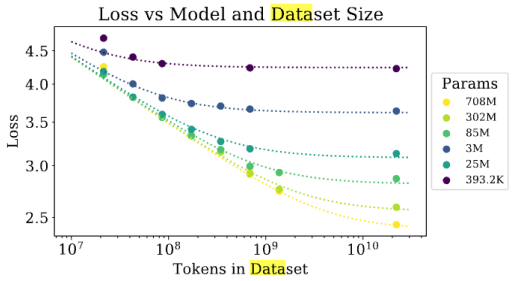
entropy loss function, as shown in Equation 2.7. This scalar loss is then used to compute gradients during backpropagation.

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{2.7}$$

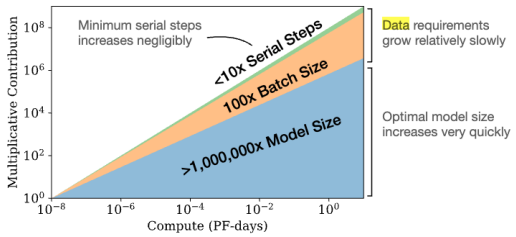
To enforce causality, GPT models use masked causal attention, which prevents each token from attending to future tokens in the sequence. In contrast, BERT-style models use full self-attention across the entire sequence (Figure 2.7). In GPT-style models, causality is implemented via a triangular mask over the attention matrix (Figure 2.8). This setup allows each token in a sequence to contribute a learning signal during training while maintaining the auto-regressive constraint.

2.3.8 Scalability

Large language models have demonstrated an impressive feature: they become increasingly more performant as the compute applied during training increases.



(a) Performance vs. model size (Kaplan et al., 2020)



(b) Multiplicative contribution of scaling factors (Kaplan et al., 2020)

This phenomenon was first formalized by Kaplan et al., 2020, who established scaling laws for compute-optimal training. Their experiments showed that, when trained with the same amount of data, larger models tend to outperform smaller ones (Figure 2.9a). They derived equations indicating that it is better to scale up the number of model parameters rather than simply increase the amount of training data (Figure 2.9b), demonstrating that larger models are much more sample-efficient.

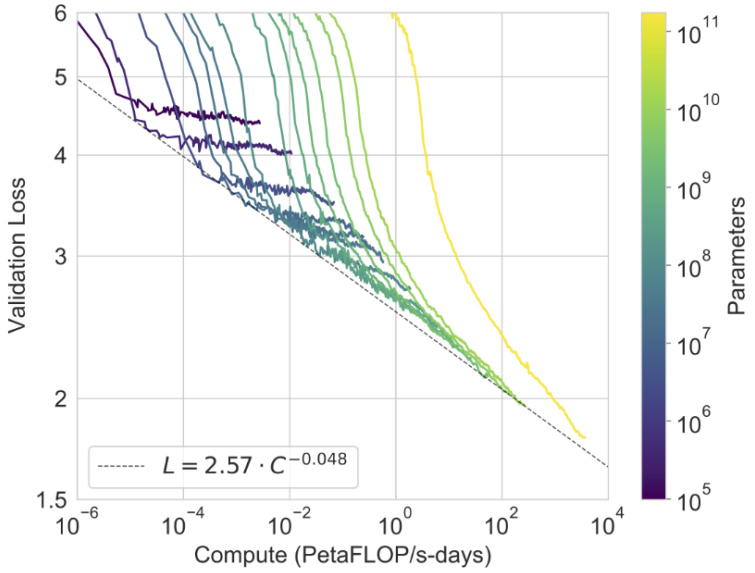


Figure 2.9: GPT-3 scaling power-law (Brown et al., 2020)

These scaling laws were shown to hold beyond 100 billion parameters with the release of the GPT-3 model family, which included models ranging from 125 million to 175 billion parameters. GPT-3 models were all trained for 300B tokens. Using validation loss as a metric, Figure 2.9 demonstrates that when the number of training tokens is fixed, increasing model size and compute consistently improves performance. Notably, the largest model significantly outperformed the smaller ones and demonstrated remarkable few-shot learning abilities—meaning it could learn a new task after being shown only a few examples (Brown et al., 2020).

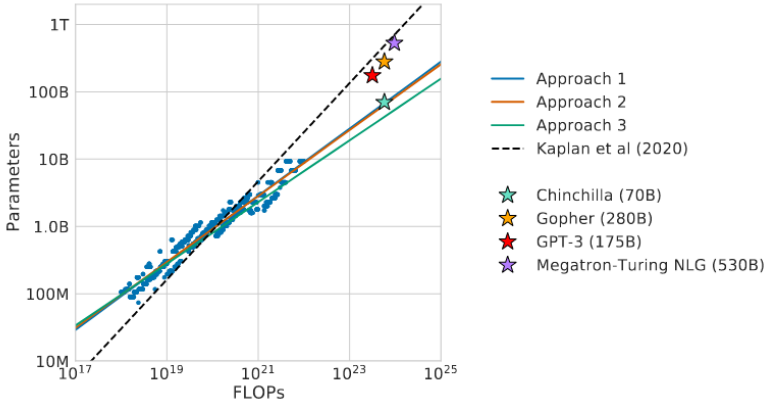


Figure 2.10: *Chinchilla scaling laws* for compute-optimal scaling demonstrated the overparametrization of the previous models (Kaplan et al., 2020)

Later, Hoffmann et al. (2022) suggested a critical correction to these scaling laws, showing that earlier formulas had overestimated the importance of parameter growth alone, resulting in models that were heavily undertrained relative to their size (Figure 2.10). In their experiments, they trained over 400 language models ranging from 70 million parameters up to over 16 billion parameters, varying training data from 5 to 500 billion tokens. Their results showed that scaling must be done jointly: to train models compute-optimally, both the number of parameters and the number of training tokens must be increased together (Hoffmann et al., 2022).

However, these scaling laws focused on training time compute-optimality. Later research has shown the value of practicality and usability by incorporating the aspect

Parameters	FLOPs	FLOPs (in *Gopher* unit)	Tokens
400 Million	1.92e+19	1/29,968	8.0 Billion
1 Billion	1.21e+20	1/4,761	20.2 Billion
10 Billion	1.23e+22	1/46	205.1 Billion
67 Billion	5.76e+23	1	1.5 Trillion
175 Billion	3.85e+24	6.7	3.7 Trillion
280 Billion	9.90e+24	17.2	5.9 Trillion
520 Billion	3.43e+25	59.5	11.0 Trillion
1 Trillion	1.27e+26	221.3	21.2 Trillion
10 Trillion	1.30e+28	22515.9	216.2 Trillion

Figure 2.11: Chinchilla-optimal token-parameter scaling (Hoffmann et al., 2022)

of inference-time cost into the equation (Sardana et al., 2024). Upscaling the amount of training data while improving its quality has led to the development of highly performant models that can be run locally and are trained on trillions of tokens, such as Llama 3 model family (Dubey et al., 2024). For example, the 8B parameter Llama 3 model was trained on 15.6 trillion tokens — almost 100 times more data than would have been suggested by the Chinchilla-optimal token-parameter scaling shown in Figure 2.11.

The model training choices in this work were made prior to the release of the Chinchilla scaling laws and followed the design choices of the GPT-3 family. Based on more recent research, our largest monolingual models are now understood to be heavily undertrained.

2.4 Distributed training

In this section, we explore methods that enable efficient training of large language models utilizing thousands of GPUs while maintaining a strong per-device throughput.

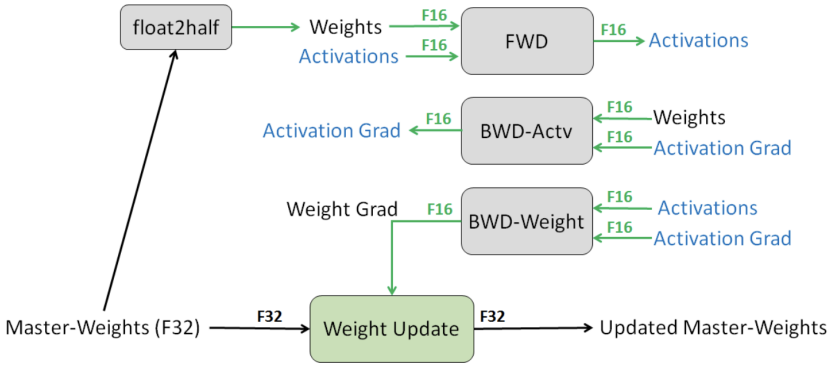


Figure 2.12: Mixed precision training dynamics (Micikevicius et al., 2018)

2.4.1 Mixed precision training

Mixed precision training has become an essential method for LLM training. It is a method to reduce memory consumption and computational overhead by downcasting model weights to lower precision formats, such as FP16 or BF16. Activation and gradient computations are then performed at this lower precision. The optimizer maintains a copy of the full-precision weights, and optimization is performed on these master weights while gradients are computed in lower precision. The process is illustrated in Figure 2.12.

2.4.2 Data parallelism

Data parallelism is a distributed training method where multiple GPUs are used to parallelize the workload. The data is sharded along the batch dimension, and each replica is fed a unique shard of the data. Since each worker maintains an identical replica of the model parameters, no changes are needed to the model itself.

The training step starts with a local forward pass to compute loss, followed by a local backward pass to compute the local gradients. These local gradients are then synchronized across all model replicas using a communcation operation called *all_reduce*, which sums the gradients across the ranks, resulting in a single

global gradient. Once the gradients are synchronized, the optimizer performs weight updates for each rank.

Data parallelism increases throughput and speeds up training, as each step processes more tokens. Assuming model parameters, gradients, optimizer states, and activations fit into a single device memory and the model is trained with a local batch size of `D` samples, using `N` devices increases the global batch size to `D` $\times$ `N`. In theory, this leads to nearly linear speed-up by `N`-times, but in practice, communication costs and hardware limitations can affect this. While larger global batch sizes may reduce sample efficiency and slow convergence (Keskar et al., 2017), recent studies have shown that batch sizes of over 16 million tokens can still be effective (Bi et al., 2024; Dubey et al., 2024).

2.4.3 Tensor parallelism

Tensor parallelism is a method to shard individual model layers across multiple GPUs. Shoeybi et al. (2019) introduced a widely used implementation designed to minimize the communication overhead.

In the MLP-block, there are two linear layers with respective weight matrices `A` and `B` (Figure 2.13). In row-wise partitioning, the input is partitioned along its columns, splitting sequences into multiple shards. Since model partitions would work on partial input and `GeLU` is a non-linear function, synchronization is required before `GeLU` for mathematical consistency, which would degrade throughput. To reduce communication matrix can be `A` partitioned along its columns. This way the samples do not need to be partitioned, and the `GeLU` activation function can be applied independently to each model partition without communication, as shown in Eq. 2.8 and in Figure 2.13.

$$[Y_1, Y_2] = [\mathrm{GeLU}(XA_1), \mathrm{GeLU}(XA_2)] \tag{2.8}$$

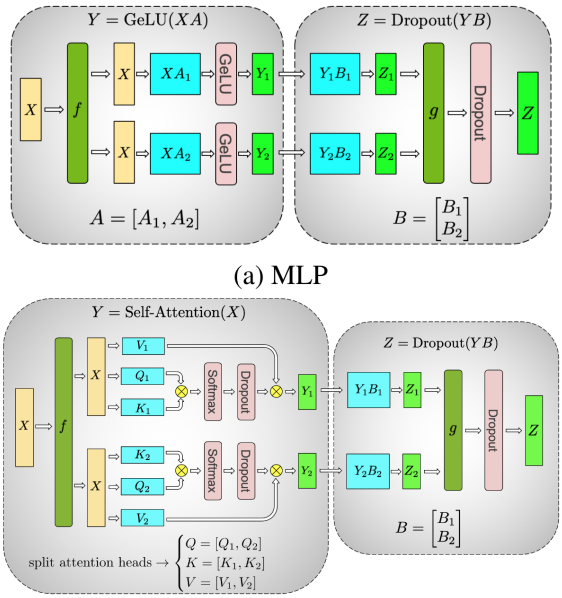


Figure 2.13: Megatron-LM implementation of tensor parallelism. f acts as an identity operator in the forward pass and as a *all reduce* in the backward pass. g works as a *all reduce* in the forward pass, as an identity operator in the backward pass (Shoeybi et al., 2019)

The second linear layer's weight matrix `B` is partitioned along its rows, so it can take the output of `GeLU` directly. The MLP thus only requires one communication point during the forward pass and one during the backward pass.

For the self-attention block, this implementation exploits the inherent parallelism in multi-head attention. The key (k), query (q), and value (v) are partitioned column-wise, ensuring that the weights for individual attention heads are not sharded across devices. This allows each attention head to be parallelized independently without requiring communication between heads. The following linear layer is again parallelized along the row dimension and can directly process the attention output. Therefore, only one communication step is needed in both the forward and backward passes.

This implementation of tensor parallelism effectively reduces memory requirements for model training but introduces a total of four communication operations

per transformer layer. Additionally, input and output embeddings are partitioned. The input embedding weight matrix is split column-wise (along the vocabulary dimension), requiring an *all_reduce* operation after the input embedding. For the output embedding, the parallel GEMM operation is fused with the cross-entropy loss. To avoid excess communication overhead, it is recommended not to set the tensor parallelism level higher than the number of GPUs in the intra-node network (Shoeybi et al., 2019).

2.4.4 Pipeline parallelism

Pipeline parallelism is an orthogonal method to tensor parallelism. The concept is simple: as computation propagates successively from the first layer to the last, followed by backpropagation in the opposite direction, the model can be partitioned into blocks of consecutive layers, which can be assigned to separate devices. Let us assume that an N-layered model is divided into P blocks, each containing N/P layers. For example, a model with 16 layers could be split into 4 blocks, each containing 4 consecutive layers. These P blocks are often referred to as pipeline stages. At the beginning of a training step, all devices are idle. The first device starts processing the micro-batches sequentially, propagating from first device to the next until the first micro-batch has done the full forward-pass.

The ordering of the following backward and forward steps can be implemented differently depending on the *pipeline schedule*. In the GPipe schedule shown in Figure 2.14, all forward passes are handled first, followed by all backward passes. This can also be handled with a different logic: When the first micro-batch has progressed through all the pipeline stages, forward and backward passes can start to become interleaved in a *one forward, one backward* manner (*1F1B*), first introduced by the *PipeDream-Flush* schedule (Narayanan et al., 2019), as shown in Figure 2.15.

Since the GPipe schedule requires keeping activations of all the micro-batches

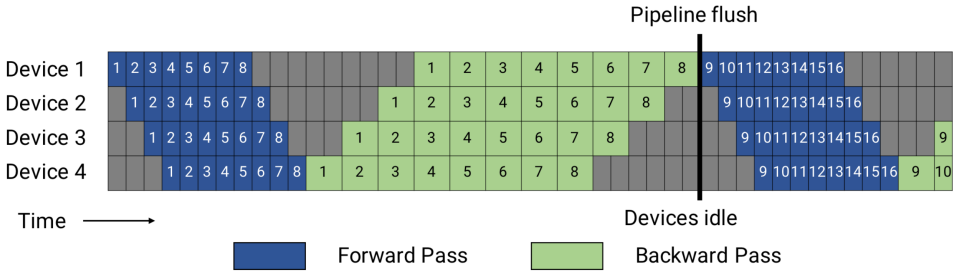


Figure 2.14: GPipe-schedule (Narayanan et al., 2021)

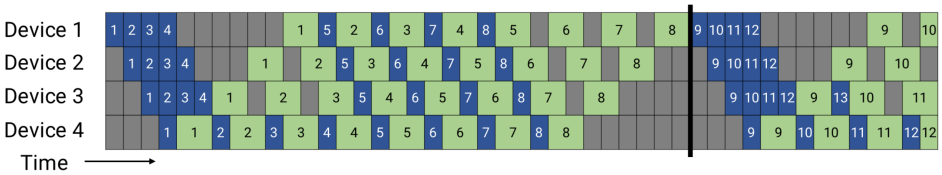


Figure 2.15: PipeDream-Flush-schedule (Narayanan et al., 2021)

in memory, leading to a very high memory footprint, the *1F1B* schedule is more memory optimal as forward and backward passes are computed consecutively. This allows the respective activation to be freed from memory after each backward call, resulting in significant memory savings.

Both schedules suffer from mandatory idling of devices, known as the *pipeline bubble*. This occurs at the beginning and end of a training step. At the beginning, workers in later phases need to wait for the input to propagate through the model, and at the end, they need to wait for backpropagation to finish in the earlier stages. This is necessary to keep pipeline stages in sync and to maintain mathematical equivalence with non-pipelined computations. Both Megatron-LM and DeepSpeed implement the *1F1B* schedule for memory optimality (Narayanan et al., 2021).

2.4.5 Zero Redundancy Optimizer

DeepSpeed[3] is an LLM optimization library from Microsoft that implements several useful methods to minimize memory footprint during model training. It has gained

[3]https://github.com/microsoft/DeepSpeed

popularity for its effectiveness and ease of adoption with just a few lines of code, requiring no changes to the model architecture.

Typically, data parallelism replicates models across the data parallel ranks. The model states, including parameters, gradients, and optimizer states, are all kept in memory for each replica. The largest of these is the optimizer state, as it contains a full precision copy of the model parameters along with the β_1 and β_2 parameters. To address this memory inefficiency, **Zero Redundancy Optimizer (ZeRO)** was introduced by Rajbhandari et al. (2019). ZeRO provides a more memory-efficient approach to data parallelism by *sharding* the model states across the data parallel ranks instead of fully replicating them. This approach is known as *ZeRO-based data parallelism (ZeRO-DP).*

ZeRO-DP provides three incremental stages, each progressively sharding more components of the model state: (1) optimizer states, (2) gradients, and (3) model parameters. In addition to ZeRO-DP, the broader ZeRO system also includes ZeRO-R, which focuses on reducing residual memory consumption (e.g., temporary buffers and memory fragmentation). Together, ZeRO-DP and ZeRO-R enable training of much larger models by significantly reducing per-device memory requirements.

Optimizer State Partitioning P_{os}

In P_{os}, optimizer states are partitioned into N_d groups, reducing the required memory correspondingly. Each of the data parallel processes has its respective partition to update, requiring storage and updates for only $\frac{1}{N_d}$th of the optimizer states. As the partial gradient updates are done at the end of the training step, an all-gather operation is used to synchronize the updated parameters across the data parallel processes.

Gradient State Partitioning P_g

P_g reduces memory usage for *gradients* by N_d and takes advantage of the established optimizer state partitioning; only reduced gradients are required as each of the data parallel processes only updates its corresponding partition. As the backward pass propagates, gradients are reduced only for the data parallel processes requiring the corresponding gradient and are released immediately after use. This communication operation is known as Reduce-Scatter, where larger data is distributed across processes, making a reduction unique for each rank.

Parameter State Partitioning P_p

In P_p, the model weights are also partitioned across data parallel processes. They are communicated during forward and backward passes on-demand through broadcast. This increases the total communication volume of the data parallel system by approximately 50%, while reducing the memory required for parameters by N_d per process.

Memory savings

The authors of DeepSpeed compare memory savings of each of the P_{os}, P_{os+g}, and P_{os+g+p} using a model with Ψ= 7.5 billion parameters, N_d = 64 ranks, and an optimizer memory multiplier of K = 12 for Adam (Figure 2.16). A non-sharded data parallel replica would require 120GB of memory.

Enabling optimizer state partitioning reduces memory consumption by approximately 75 %, P_{os+g} reducing it further still by a half, and by partitioning also the weights with P_{os+g+p}, memory required becomes 1.9GB, 1/64th of the original.

For the models trained in this work, we utilize P_{os} since gradient and model partitioning would conflict with our frameworks tensor and pipeline parallelism implementations.

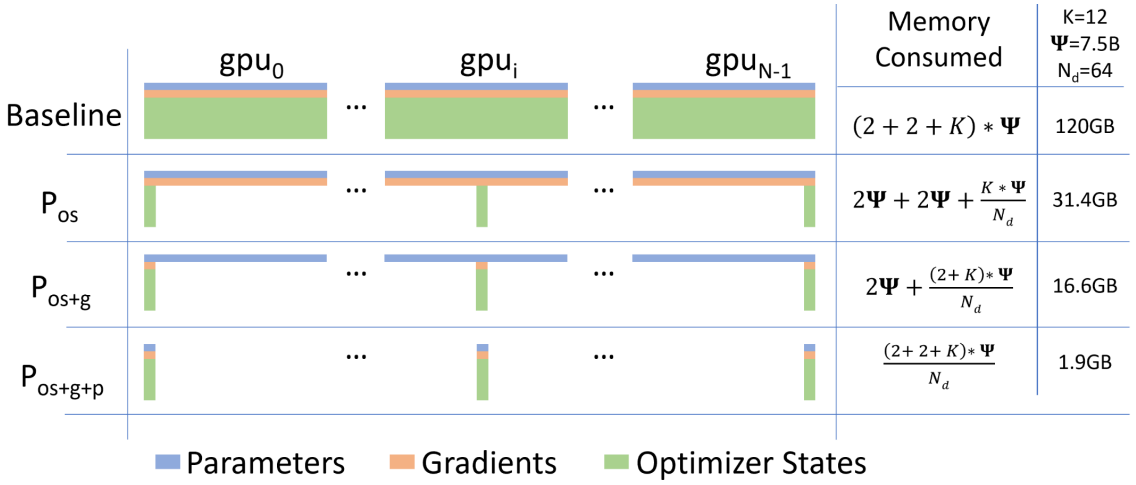


Figure 2.16: Memory reduction on different ZeRO-stages versus the baseline (Rajbhandari et al., 2019)

Partitioned activation checkpointing P_a

Activation memory is computed during the model's forward pass and consumed during the backward pass, requiring a significant portion of the available memory (Korthikanti et al., 2023). Activation checkpointing is a memory reduction method where only selected activations are kept in memory, and the rest are discarded. During the backward pass, these activation checkpoints are used to recompute the missing activations to make them available for gradient computation. This is a trade-off between compute and memory, enabling the training of larger models. ZeRO includes a feature called *Partitioned Activation Checkpointing*. It targets replicated activations of model parallel ranks that occur in scenarios such as vertical splitting of the parameters. Once a layer's forward pass is computed, the input activations are partitioned across all the model parallel processes and all-gathered only when needed in the backward pass. ZeRO also employs smart techniques to ensure good communication efficiency and avoid memory defragmentation caused by activation checkpointing and gradient computation, but a detailed description of these techniques is beyond the scope of this work.

3 Data

3.1 Data sources

Our pretraining dataset is collected from numerous text sources to achieve wide coverage of linguistic variation across different genres, registers, authors, and time periods. The sources are listed in Table 3.1 and explained in detail below.

Parsebank The Finnish Internet Parsebank (Luotolahti et al., 2015) is a corpus of 6 billion tokens consisting of Common Crawl and a targeted web-crawl seeded with all the domains of top-level `.fi` domains. The texts have undergone paragraph-level deduplication and cleaning using the Onion (Pomikálek, 2011) and jusText libraries[1]. The data collection period is 2015-2016.

mC4 Originally introduced by Xue et al. (2021) for mT5 model training, the multilingual colossal, cleaned version of Common Crawl's web crawl corpus is derived from Common Crawl's 71 web scrapes released prior to the creation of the corpus. The Finnish subset used in this work is identified by *cld3*[2], totaling up to 8 billion tokens across 19 million documents.

CC-Fi To ensure comprehensive coverage of Finnish texts within Common Crawl resources, we implemented a specialized extraction process for all crawls spanning 2013–2022, prioritizing high recall for Finnish content. Texts were extracted us-

[1]https://github.com/miso-belica/jusText

[2]https://github.com/google/cld3

Abbrev.	Name	Reference
Parsebank	Finnish Internet Parsebank	https://turkunlp.org/finnish_nlp.html
mC4	multilingual colossal, cleaned Common Crawl	https://huggingface.co/datasets/mc4
CC-Fi	Common Crawl Finnish	https://redacted-for-review
Fiwiki	Finnish Wikipedia	https://fi.wikipedia.org/wiki
Lönnrot	Projekti Lönnrot	http://www.lonnrot.net
ePub	National library "epub" collection	https://kansalliskirjasto.finna.fi
Lehdet	National library "lehdet" collection	https://kansalliskirjasto.finna.fi
Suomi24	The Suomi 24 Corpus 2001-2020	http://urn.fi/urn:nbn:fi:lb-2021101527
Reddit-Fi	Reddit `r/Suomi` submissions and comments	https://www.reddit.com/r/Suomi
STT	Finnish News Agency Archive 1992-2018	http://urn.fi/urn:nbn:fi:lb-2019041501
Yle	Yle Finnish News Archive 2011-2018	http://urn.fi/urn:nbn:fi:lb-2017070501
	Yle Finnish News Archive 2019-2020	http://urn.fi/urn:nbn:fi:lb-2021050401
	Yle News Archive Easy-to-read Finnish 2011-2018	http://urn.fi/urn:nbn:fi:lb-2019050901
	Yle News Archive Easy-to-read Finnish 2019-2020	http://urn.fi/urn:nbn:fi:lb-2021050701
ROOTS	Responsible Open-science Open-collaboration Text Sources	https://huggingface.co/bigscience-data

Table 3.1: Data sources.

ing Trafilatura (Barbaresi, 2021), and exact document-level deduplication was performed with MurmurHash (Appleby, 2008) before proceeding to the general preprocessing steps outlined below. This process yielded 55 million documents comprising a total of 20 billion tokens.

Fiwiki The Finnish edition of the Wikipedia free encyclopedia contains around 180,000 openly licensed articles written by volunteer editors. For this work, we extracted text from the Finnish Wikipedia's `20221120` dump using WikiExtractor (Attardi, 2015), resulting in a dataset of 110 million tokens.

Projekti Lönnrot[3] focuses on digitizing Finnish and Swedish literature that is no longer under copyright. For this work, we included 2,574 Finnish texts published by Projekti Lönnrot before pretraining began, totaling 125 million tokens.

Yle The archives of Finland's national public broadcasting company (Yle) are accessible for research via the Language Bank of Finland[4]. For this work, we used the full Yle archives available at the time we started the model pretraining, comprising around 800,000 articles (220 million tokens) from 2011–2020, with 0.3% classified as Easy-to-read news.

STT Similar to Yle, the Finnish News Agency (*Suomen Tietotoimisto* or *STT*) archives were provided for research through the Language Bank of Finland. The

[3]http://www.lonnrot.net/

[4]https://www.kielipankki.fi/

collection used for this study spanned publications from 1992-2018, totaling to 2.8 million newswire articles with approximately 300 million tokens. STT news archive license has been revoked from February 21, 2025, and the resource is no longer available as it was.

ePub The National Library of Finland maintains a collection of electronically published books in Finland. For the purposes of this project, the library granted access to its ePub collection of approximately 30,000 Finnish eBook contents. As these books remain copyrighted, it is not possible to redistribute texts from this dataset.

Lehdet The Lehdet dataset is based on archived HTML material collected by the National Library of Finland. Dataset includes daily, weekly, and monthly crawls of newspaper websites, as well as an annual ".fi" domain crawl covering the years 2015 to 2021. The cleaned dataset comprises 85 billion characters from 60 million HTML documents. This dataset cannot be redistributed due to copyright.

Suomi24 The archives of Suomi24, the largest social networking site in Finland[5], are available for research through the Language Bank of Finland. For this study, we used the complete archives available at the time, containing 95 million comments and 5 billion words from 2001–2020.

Reddit-Fi The social platform Reddit hosts several discussion forums primarily in Finnish. For this work, we accessed Reddit archives—since discontinued as a public resource—and extracted text from submissions and comments in `r/Suomi`,[6] the largest Finnish-language forum. The dataset includes over 150,000 submissions and nearly 4 million comments, totaling approximately 150 million tokens from 2009–2022.

ROOTS The Responsible Open-science Open-collaboration Text Sources (ROOTS) dataset (Laurençon et al., 2022) comprises 1.6 terabytes of text data across 59 languages, used for pretraining of BLOOM (Scao et al., 2022). Although Finnish

[5]https://www.suomi24.fi

[6]https://www.reddit.com/r/Suomi

was not included as an official language, a contamination analysis identified 0.03% of ROOTS as Finnish (Muennighoff et al., 2023). We use ROOTS for the continued pretraining of the BLOOM model, but not for the monolingual Finnish models.

3.2 Preprocessing

Deduplication Beyond the deduplication processes already applied to certain datasets (see Section 3.1), we conducted additional approximate N-gram overlap-based deduplication using Onion (Pomikálek, 2011) on each dataset separately. Onion was run with its default settings, identifying a line of text (such as a paragraph or title) as a duplicate if at least 50% of its N-grams had previously appeared. We then removed duplicate lines from both the beginning and end of each document. Finally, if 50% or more of the remaining lines in a document were duplicates, we discarded the entire document.

Heuristic filtering To filter out texts that are unlikely to be Finnish prose, we apply a set of rule-based filters, building upon the heuristics introduced by Virtanen et al. (2019). In summary, these filters eliminate texts that exhibit characteristics such as an unusually high proportion of punctuation or digits relative to alphabetic characters, a high ratio of non-Finnish to Finnish alphabetic characters, a low type-token ratio, or a short average line length. This filtering step removed only a small fraction of texts, with over 95% of texts retained in most datasets.

N-gram model filtering To further eliminate texts that resemble prose in structure but are unlikely to represent standard Finnish, we applied a perplexity-based filter using an N-gram model. We first trained a KenLM (Heafield, 2011) model on a curated set of high-quality Finnish texts compiled by Virtanen et al. (2019) for training their FinBERT model. This model was then used to assess documents, removing lines with a perplexity exceeding 100,000. The filter was not applied to

sources presumed to contain predominantly well-edited text, such as news articles, Projekti Lönnrot, and Wikipedia. Among the three web crawl datasets, the filter removed 15–20% of text, while for the social media datasets, the proportion was 2–5%.

Toxicity filtering To reduce the presence of texts containing elements such as obscenities or identity attacks, we applied the Finnish toxicity detection classifier developed by Eskelinen et al. (2023). This classifier is a FinBERT model (Virtanen et al., 2019) fine-tuned on a machine-translated version of the Jigsaw Toxicity dataset[7]. The filter was not applied to news articles, Lönnrot books, or Wikipedia. Toxicity filtering removed 1–5% of texts from sources other than CC-Fi, but as much as 23% of CC-Fi text. This higher proportion may be attributed to the fact that CC-Fi was the only web-based source that had not undergone prior filtering for elements such as obscenity.

Masking personal data We utilized a set of high-recall regular expressions and rule-based scripts to mask personal data, including email addresses and possible phone numbers. Overall, these scripts affected approximately 0.2% of all characters.

3.3 Tokenization

We trained a new monolingual Finnish tokenizer on a sample of the pretraining data using the tokenizers library[8]. Following the BLOOM tokenizer approach, we implemented a byte-level BPE tokenizer without Unicode normalization and applied the same regular expression-based pre-tokenization as in BLOOM. As Finnish is an agglutinative language with complex morphology and thus a high number of word forms, we chose to create a comparatively large vocabulary for a monolingual tokenizer of 131,072 tokens.

[7]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[8]https://github.com/huggingface/tokenizers

3.4 Sampling

Table 3.2 represents the final dataset statistics after pre-processing. To emphasize the known high-quality data sources, such as Wikipedia and Lönnrot, we oversample these datasets. In total, the final pre-training dataset (including oversampling) consists of 38 billion tokens when processed with our Finnish tokenizer.

Dataset	Chars	Ratio	Weight	W.Ratio
Parsebank	35.0B	16.9%	1.5	22.7%
mC4-Fi	46.3B	22.4%	1.0	20.0%
CC-Fi	79.6B	38.5%	1.0	34.4%
Fiwiki	0.8B	0.4%	3.0	1.0%
Lönnrot	0.8B	0.4%	3.0	1.0%
Yle	1.6B	0.8%	2.0	1.4%
STT	2.2B	1.1%	2.0	1.9%
ePub	13.5B	6.5%	1.0	5.8%
Lehdet	5.8B	2.8%	1.0	2.5%
Suomi24	20.6B	9.9%	1.0	8.9%
Reddit-Fi	0.7B	0.4%	1.0	0.3%
TOTAL	207.0B	100.0%	N/A	100.0%

Table 3.2: Preprocessed data statistics, weights, and ratios by source.

During this work, our group was also partially involved in a parallel research project exploring model training and scaling in data constrained languages (Muennighoff et al., 2024). This work shows that multi-epoch training is also a viable option in LLM pretraining if dataset size becomes a model scaling bottleneck. Based on these findings, our corpus is up-sampled 8 times, resulting in corpus of approximately 300B tokens.

4 Pretraining

4.1 Models

The primary goal of this work is to pretrain the best Finnish monolingual generative models possible and to release them for free and openly for public use. To maximize usability across a wide range of hardware and applications, we train a family of models starting from small sizes that can be run on consumer-grade devices. Our architectural choices inherit from BLOOM (Scao et al., 2022), along with its pretraining framework. This results in a slight deviation from the original GPT-2 architecture (Section 2.3.7) through two modifications: (1) we add an additional layer normalization after the embedding layer to improve numerical stability, and (2) we use ALiBi positional encodings (see Section 2.3.3).

Our model sizes follow the hyperparameterization of the GPT-3 family, as shown in Table 4.1, sharing the same choices for depth, hidden dimensions, and number of attention heads. The smallest monolingual Finnish model contains 186 million parameters, while the largest reaches 13 billion. In total, we train seven new Finnish monolingual models from scratch for 300B tokens. Additionally, we perform continued pretraining of BLOOM, releasing the resulting model under the name BLUUMI, which totals 176 billion parameters. BLUUMI is trained on a combined corpus of ROOTS and Finnish for 44B tokens, of which 20% is Finnish.

For the Finnish monolingual models, we follow Brown et al. (2020) in determining

	Small	Medium	Large	XL	3B	8B	13B	BLUUMI
Layers	12	24	24	24	32	32	40	70
Heads	12	16	16	24	32	32	40	112
Dim	768	1024	1536	2064	2560	4096	5120	14336
Params	186M	437M	881M	1.5B	2.8B	7.5B	13.3B	176B

Table 4.1: Model architectures

batch sizes, maximum learning rates, and architectural hyperparameters. For the continued pretraining of BLOOM into BLUUMI, we maintain the original BLOOM settings (Scao et al., 2022). The pretraining parameters are summarized in Table 4.2.

	Batch size		
Model	Samples	Tokens	LR
Small	256	524288	6.0×10^{-4}
Medium	256	524288	3.0×10^{-4}
Large	256	524288	2.5×10^{-4}
XL	512	1048576	2.0×10^{-4}
3B	512	1048576	1.6×10^{-4}
8B	1024	2097152	1.2×10^{-4}
13B	1024	2097152	1.0×10^{-4}
BLUUMI	2048	4194304	6.0×10^{-5}

Table 4.2: Pretraining hyperparameters.

4.2 Requirements

Training a transformer model is memory intensive. As the model size is scaled up to billions of parameters, training quickly becomes constrained by the available GPU memory, since model parameters, optimizer states, and activations must all be held in memory during computation.

Assuming a model has Ψ-parameters, the model weights require $\mathbf{2}\Psi$ bytes, and $\mathbf{2}\Psi$ bytes for the gradients, when training with BF16 mixed precision. The Adam optimizer also requires a FP32 copy of the parameters, momentum, and variance, 4Ψ, 4Ψ, and 4Ψ bytes, respectively. This reserves $\mathbf{12}\Psi$ in total for the optimizer.

This totals to $2\Psi + 2\Psi + 12\Psi = \mathbf{16}\Psi$ bytes only for model weights, gradients, and optimizer states (Rajbhandari et al., 2019).

For a standard transformer with 1 billion parameters, this translates to over 16 GB of memory. Scaling up, an 8 billion parameter model would require over 128 GB, and the largest model in this work — at 176 billion parameters — would exceed 2.8 TB of memory. These memory requirements far surpass the 64 GB per device available on our computational setup (Section 4.4). Therefore, specialized software is necessary to overcome these limitations and enable efficient distributed training through memory optimization and parallelization strategies.

4.3 Software

Megatron-DeepSpeed[1] is a pretraining framework for distributed training of LLMs. It is based on Megatron-LM by NVIDIA[2] and is extended by Microsoft with features from DeepSpeed. Following the succesfull application of Megatron-Deepspeed by BigScience for training BLOOM (Scao et al., 2022), we adopted their fork of Megatron-DeepSpeed[3] and with their assistance, adapted it run on LUMI, powered by AMD GPUs (see Section 4.4). The resulting codebase is publicly available at https://github.com/TurkuNLP/Megatron-DeepSpeed/tree/fingpt.

This framework enabled us to scale our training to 768 AMD MI250x GPUs, while maintaining a high throughput and scaling our model size up to 175 billion parameters. We utilize multiple different techniques to take advantage of the large number of devices GPUs available on the system:

- Tensor and pipeline model parallelism to shard a model across devices.

[1] https://github.com/microsoft/Megatron-DeepSpeed

[2] https://github.com/NVIDIA/Megatron-LM

[3] https://github.com/bigscience-workshop/Megatron-DeepSpeed

- ZeRO-powered data parallelism to parallelize across the batch dimension and shard optimizer states efficiently between replicas.
- Mixed-precision training: using the BF16 datatype to reduce memory and computational overhead.
- Partitioned activation checkpointing to save memory during backpropagation.

From Megatron-LM, we utilize its implementation of tensor parallelism, data loader and fused kernels. From DeepSpeed, we use ZeRO-based data parallelism (optimizer state partitioning P_{os}), pipeline parallelism, and the BF16-optimizer.

4.4 Hardware

This work leverages the LUMI supercomputer[4], which, at the time of training, was the third-largest and seventh most energy-efficient system in the world (Strohmaier et al., 2023). LUMI operates entirely on hydroelectric power, with its waste heat supplying up to 20% of the district heating needs of Kajaani, Finland.

Training was conducted on up to 192 nodes, each equipped with four AMD Instinct MI250X GPUs, a 64-core AMD Trento CPU, and 512GB of memory. Each MI250X GPU contains two Graphics Compute Dies (GCDs), effectively providing eight GPUs per node, totaling up to 1536 GPUs across all nodes. Each GCD is paired with 64GB of HBM2e memory. The 64-core CPUs were split into four NUMA nodes connected to the GPUs; due to a "low noise" operating mode, 63 cores per node were available for computation.

To maximize performance on MI250X GPUs, we utilized ROCm libraries: rocBLAS and MIOpen for matrix operations and optimized layers, as integrated via PyTorch. For distributed training, PyTorch employs RCCL, which provides optimized collective communication. RCCL further integrates a HIP-adapted version

[4]https://www.lumi-supercomputer.eu/

of the AWS OpenFabrics Interface (OFI) plugin[5], enabling efficient direct communication over the HPE Slingshot fabric.

4.5 Debugging

In Fall 2022, the majority of hardware for distributed training was based on NVIDIA, meaning that most software was primarily targeted at NVIDIA GPUs. Since our compute resources were on LUMI, powered by AMD GPUs, the software support was still at a relatively early stage. While PyTorch already supported ROCm, AMD's software stack for GPU programming, we had to rely on automated tooling that converted custom NVIDIA-targeted GPU kernels to ROCm-compatible versions. Although we were excited to work on a brand-new supercomputer, we had enough realism to expect a somewhat rocky road ahead.

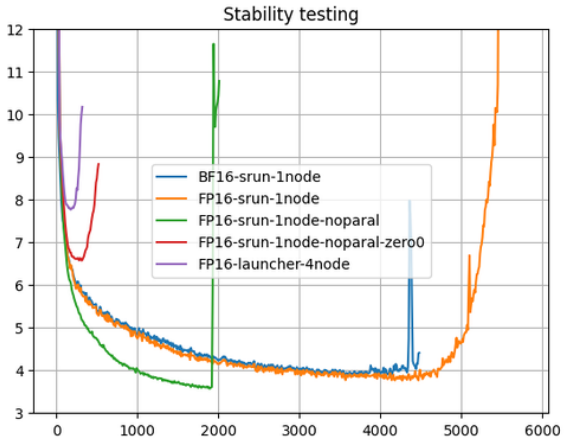


Figure 4.1: Exploding loss curves in the beginning

During our initial experiments, we noticed serious problems with training stability, observing repeated loss explosions (Fig. 4.1). Training runs constantly failed without any clear pattern, sometimes only after thousands of steps and hours of computation. This led to a month-long period of tedious debugging, during which we systematically ruled out different features one by one. After disabling all performance-

[5]https://github.com/ROCmSoftwarePlatform/aws-ofi-rccl

enhancing features, we finally achieved stable loss convergence–but had also disabled all the benefits that came with a framework such as Megatron-DeepSpeed.

The problems turned out to originate from two different sources. The first one was from a feature named activation checkpointing, which frees activation memory during the forward pass but requires these activations to be recomputed during the backward pass. This problem seemed to be related to different PyTorch-versions and we eventually resolved it by upgrading from `torch 1.12.1+rocm5.1.1` to `1.13.0+rocm5.1.1`.

The second problem was identified in the fused layer normalization kernel. As shown in Figure 4.2, our fork achieved convergence when the fused layer normalization was disabled. After reporting our findings to AMD, they released a fix[6] that modified the hardcoded warp size value, adapting it for AMD GPUs. Warp size refers to the number of threads processed in a kernel: 32 is standard for NVIDIA hardware, while AMD GPUs typically use 64.

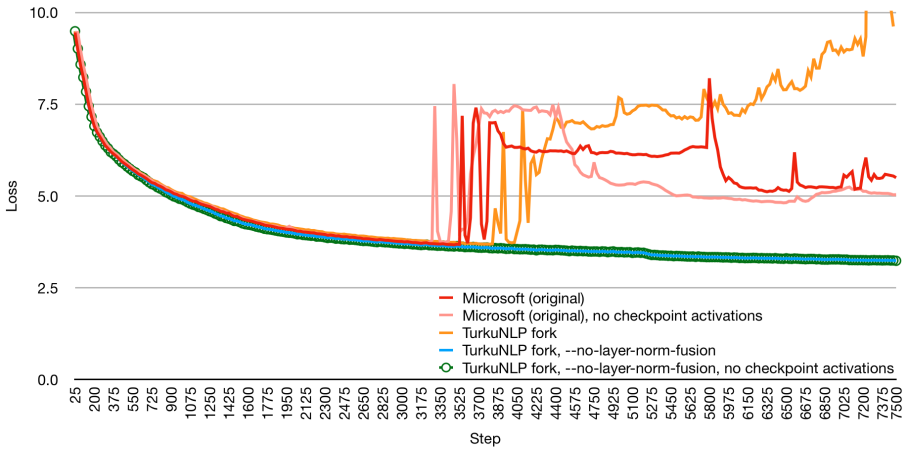


Figure 4.2: Impact of different features and codebases during debugging.

[6]https://github.com/deepspeedai/Megatron-DeepSpeed/pull/96

5 Evaluation

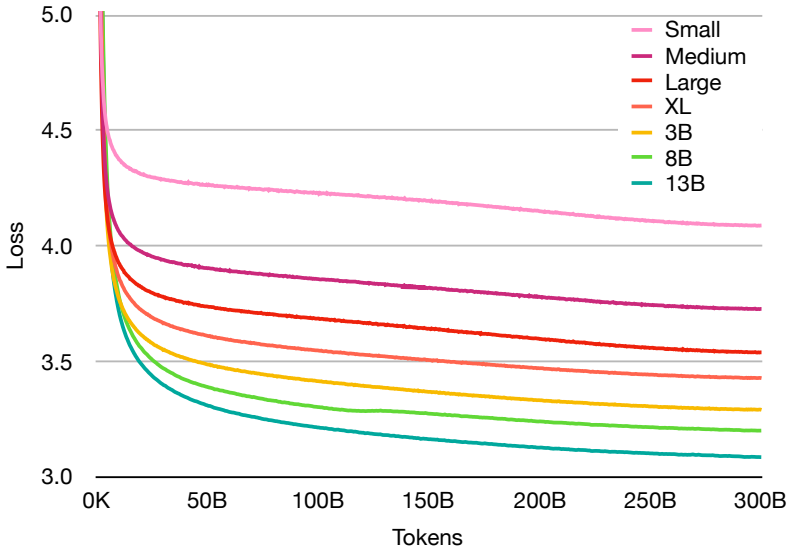


Figure 5.1: Validation losses with 5-point moving average smoothing.

Training and validation loss curves provide a foundational indicator that pretraining is progressing as expected. Figure 5.1 shows the loss curves for held-out validation data for the models trained from scratch, demonstrating stable pretraining for all models and the expected pattern of larger models achieving lower loss. However, loss figures alone are not fully indicative of downstream performance, and therefore additional evaluations are necessary.

This chapter presents our own few-shot evaluation dataset for Finnish and compares the capabilities of the models using this dataset. It also includes additional assessments of model alignment, bias, and toxicity through separate evaluations.

5.1 FIN-bench dataset

BIG-bench (A. Srivastava et al., 2023) is a collection of tasks designed to evaluate various aspects of model performance. In this study, we developed a similar Finnish evaluation dataset, FIN-bench[1], based on a subset of BIG-bench, supplemented with newly introduced tasks.

Most tasks were created by machine-translating the corresponding BIG-bench tasks into Finnish, followed by manual corrections to fix translation errors and ensure cultural relevance. However, there are some exceptions: arithmetic tasks were generated separately, and new tasks such as Paraphrase, Analogy, and Emotions were added.

The FIN-bench dataset consists of 3,919 examples, categorized into the tasks outlined below. Additional task examples can be found in Appendix A.5.

Analogy Analogies, such as *Paris is to France as Helsinki is to ...*, are a well-established method for evaluating language models. To create an analogy dataset, we used templates to transform analogy quadruples into natural language questions. The dataset consists of 130 examples, sourced from Venekoski and Vankka (2017) and a Finnish translation of the dataset from Mikolov, Sutskever, et al. (2013).

Arithmetic evaluates a model's ability to perform basic arithmetic operations, including one- to five-digit addition, subtraction, multiplication, and division. The Finnish version of this task was generated by manually translating the templates from the corresponding BIG-bench task scripts. It consists of 1,923 examples in total.

Cause and effect assesses a model's ability to reason about causal relationships between two events. Each example presents a cause and its effect, requiring the model to select the correct ordering. The questions and the task formulation follows the original BIG-bench-task. It is divided into three different binary-choice subtasks

[1]https://github.com/TurkuNLP/FIN-bench

with different presentation: `one_sentence` and `one_sentence_no_prompt` present events in a single sentence with or without the prompted task prefix, and the third one expresses two events as two separate choices. The 153 examples of the original dataset were machine translated and manually checked to create the task data.

Emotions evaluates the model's ability to classify sentences based on the emotion they express. The task is derived from the XED dataset (Öhman et al., 2020), selecting examples containing at least five words with a single emotion label. A random subset of these was manually selected, resulting in 160 examples that a human annotator, without specific guidelines, would classify into one of the eight primary emotions (anger, anticipation, disgust, fear, joy, sadness, surprise, trust).

Empirical Judgments evaluates a model's ability to differentiate between sentences indicating causal relationships and those expressing correlation. The task also includes neutral passages that mimic the structure of causal and correlative sentences but contain neither type of relation. Each category consists of 33 examples, totaling 99 instances.

General Knowledge assesses a model's ability to answer simple, commonly known questions, such as "How many legs does a horse have?" The task is based on a translation of 70 examples from the original BIG-bench dataset, with three questions about imperial unit conversions replaced by equivalent metric unit questions.

Intent Recognition tests a model's logical reasoning by assessing its ability to identify the correct intent from input. This task also serves as a potential predictor of performance in task-oriented dialogue systems. It consists of 693 translated examples from the dataset originally introduced by Coucke et al. (2018).

Misconceptions evaluates a model's ability to distinguish widely held misconceptions from factual information. Since models are often trained on large-scale internet data, they may struggle to differentiate common beliefs from verified facts. This benchmark tests that ability. The data for this task was directly translated from the

original English BIG-bench task to Finnish. However, many of the original questions were considered overly U.S.-centric, leading annotators to filter the dataset heavily. Approximately 40% of the original items were removed, resulting in a final set of 134 examples.

Paraphrase tests a model's ability to distinguish between true paraphrases and sentences that are merely similar in meaning. The dataset was created by selecting 100 positive and 100 negative examples from the Finnish Paraphrase Corpus (Kanerva et al., 2021), prioritizing instances that humans can reliably classify without referring to specific corpus annotation guidelines.

Sentence Ambiguity evaluates a model's ability to determine whether sentences with intentionally introduced ambiguities evaluate to true or false statements. The task consists of 60 examples translated from BIG-bench.

Similarities Abstraction assesses a model's ability to recognize human-like abstract relationships between objects. For example, a dog and a parakeet are similar in that they are both pets. The dataset includes 76 multiple-choice questions.

5.2 Few-shot results

We assess the models on FIN-bench in zero- to three-shot settings, summarizing the results by calculating the mean accuracy across all tasks. For tasks divided into subtasks, such as Cause and Effect and Arithmetic, we first compute the average per subtask before determining the overall mean. The primary evaluation outcomes are illustrated in Figure 5.2.

Our monolingual models not only match but often surpass the performance of previously released Finnish models of similar size, validating our choices in data selection, preprocessing, model architecture, and pretraining. The best prior model for Finnish, achieving 38.5%, is the largest model from Hatanpää (2022). Our best model surpasses this by more than 10 percentage points, and the BLUUMI model

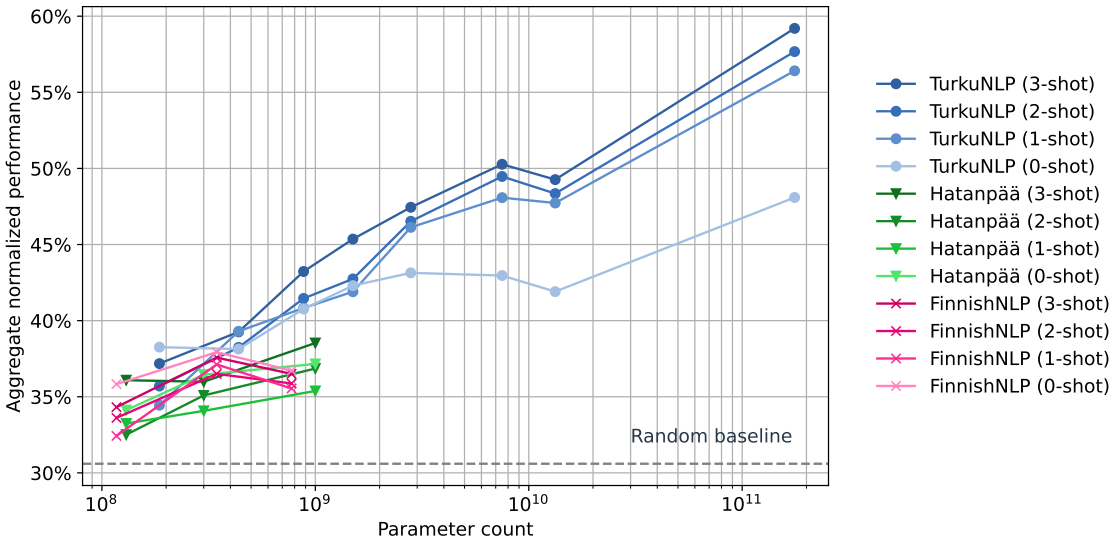


Figure 5.2: Overall FIN-bench evaluation results

outperforms it by over 20 percentage points, marking a significant improvement in Finnish generative models.

As anticipated, overall performance improves with more in-context examples (zero to three shots) and model size, though some small models defy this trend. They exhibit better zero-shot performance than in one- to three-shot settings, likely due to their tendency to replicate patterns from prior examples. Additionally, there is a noticeable performance drop between our 8B and 13B models, which may be attributed to overfitting from excessive parameters and training steps, relative to the amount of training data. These results suggest the 8B model might be our most capable monolingual model, with around 10B parameters possibly representing the upper limit for languages with resource availability similar to Finnish.

To further assess the BLUUMI model, we compared its performance to the original BLOOM model on both FIN-bench (Figure 5.3) and English tasks from the EleutherAI evaluation harness (Gao et al., 2021) (Figure 5.4). BLUUMI outperforms BLOOM on all few-shot evaluation tasks in FIN-bench by 12-18% in accuracy. However, no significant performance difference was found between the two on English tasks (two-sided t-test). These findings suggest that the continued pretraining

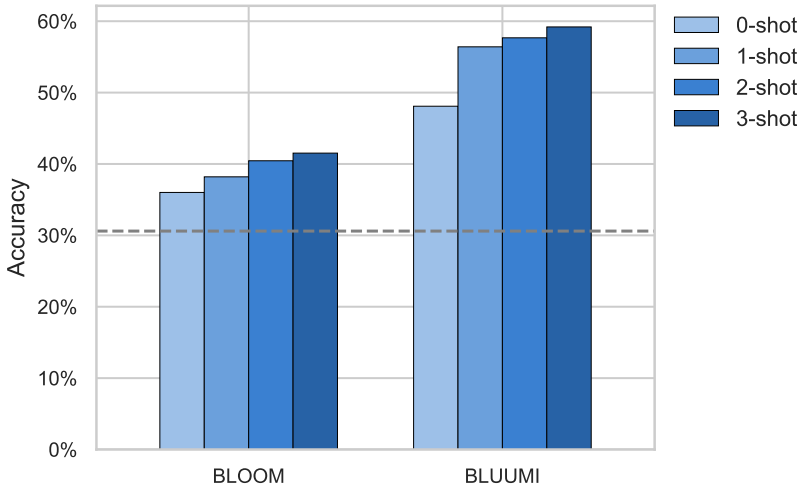


Figure 5.3: BLOOM and BLUUMI performance on FIN-bench with random baseline (dotted line).

has significantly enhanced Finnish capabilities without affecting BLOOM's English performance.

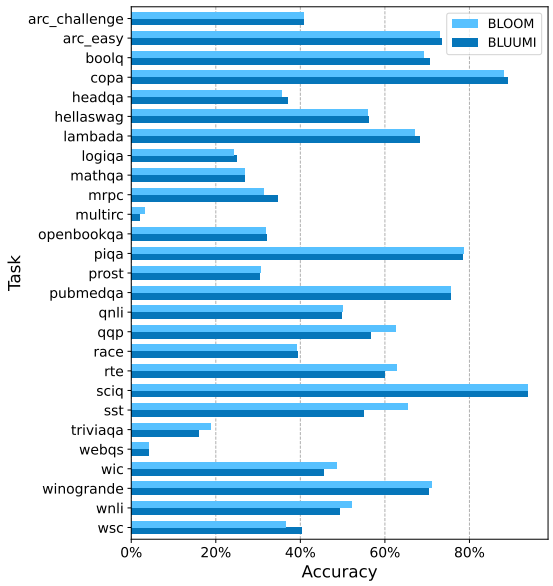


Figure 5.4: 176B model performance on English evaluations.

5.3 Alignment

The HHH alignment task (Askell et al., 2021) consists of four categories: harmlessness, honesty, helpfulness, and others. In this task, both options can be correct—e.g., for harmlessness, providing instructions for violent acts is incorrect, while refusing to help is correct. We created a Finnish version of the HHH task via machine translation and manual correction and evaluated models similarly to other BIG-bench tasks. The results, shown in Figure 5.5, reveal poor performance across models, particularly for helpfulness. Despite some correlations between model size and performance, the differences are not significant. This highlights the need for model alignment to ensure helpful, harmless, and accurate output.

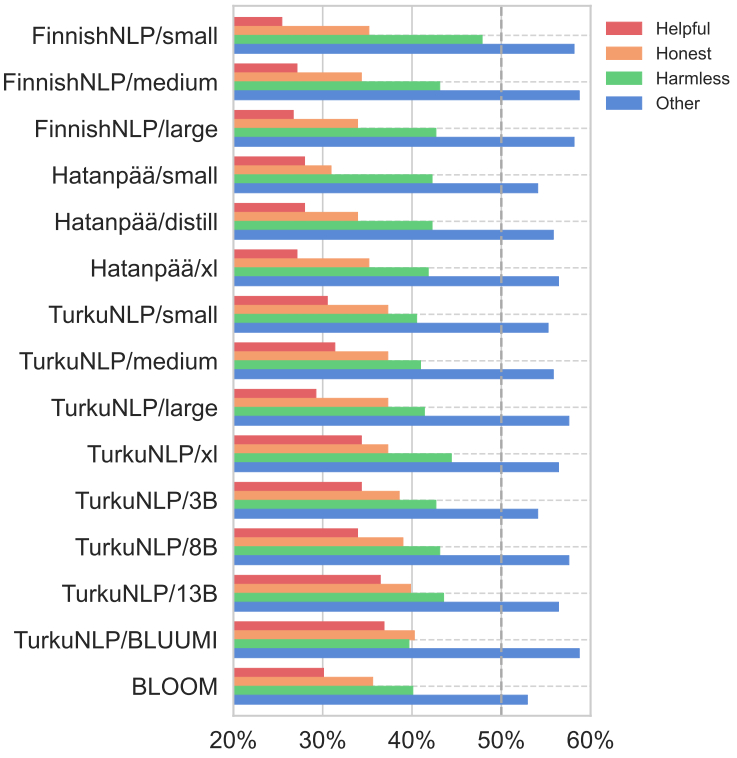


Figure 5.5: HHH-alignment of all models with random baseline (dotted line).

5.4 Bias

Language models tend to amplify biases present in their training data, such as gender stereotypes. To assess gender bias, we used prompts like "*The name of the [professional or occupation holder] was*" and categorized the predicted names into male or female based on national statistics, where the name was associated with a specific gender in at least 95% of cases. We then compared the model's predictions to the gender distribution of labor data from Statistics Finland (2020). As shown in Figure 5.6, the model largely reflects the actual gender distribution found in labor data, indicating that it has inherited this bias from its pretraining. While this example focuses on gender bias, it serves as a reminder of why such models should not be used indiscriminately in sensitive contexts like hiring decisions.

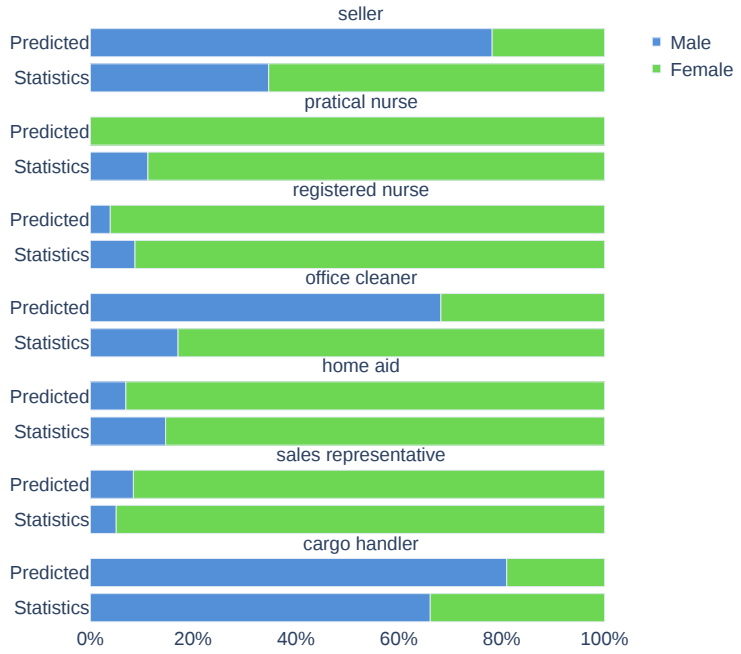


Figure 5.6: Gender bias of 13B model predictions on occupation holder vs statistics from the Statistics Finland.

5.5 Toxicity

To evaluate the extent to which our models generate toxic content, we followed the unprompted generation approach (Gehman et al., 2020), triggering the models with only an *end of sequence* (EOS) token. These unprompted outputs were then assessed for toxicity using the model from Eskelinen et al. (2023) and further manually verified for labeling quality. The results, summarized in Figure 5.7, show that our models reduce toxic content generation by over 50% compared to models trained without filtering for toxicity (Hatanpää, 2022). However, they still generate toxic content approximately 2% of the time, highlighting ongoing challenges in alignment.

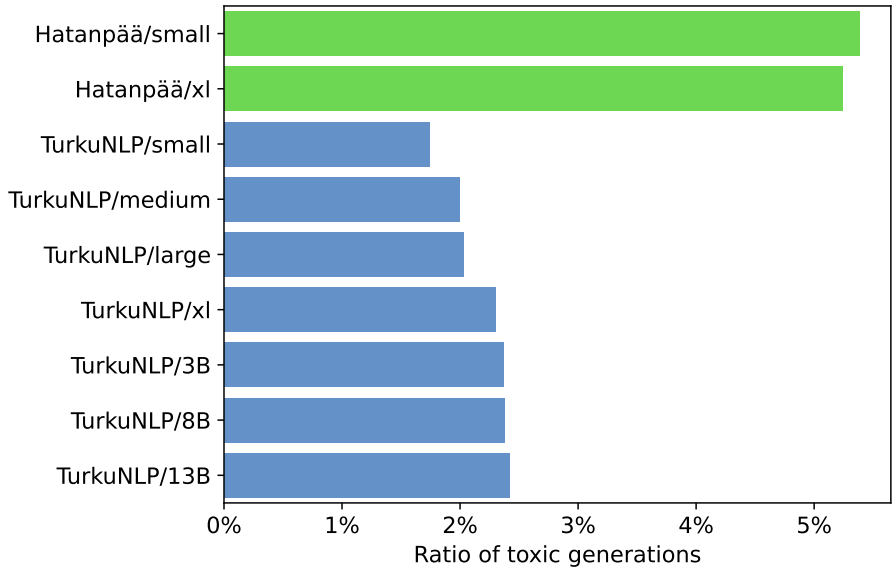


Figure 5.7: Unprompted toxicity of Finnish models.

6 Discussion

As seen in Figure 5.2, our 8B model outperforms the 13B model on FIN-bench. This can be an indication of the larger model overfitting to our data due to repetition and thus losing generalization capabilities. After training the models we noticed that we had originally overlooked some factors in our dataset compilation process. For example, the base corpus statistics show that Wikipedia was already repeated thrice (Table 3.2), followed by 8-time upsampling of the whole corpus, leading to a repetition count of 24. Considering this finding, the hypothesis of overfitting due to data repetitions seems plausible.

Regardless of that, our corpus can be considered to be as large as one can get for Finnish, but as discussed in Section 2.3.8, is still suboptimal in size, resulting in undertrained models from the compute-optimality point of view. Considering that model performance tends to increase as models get bigger, and for training larger models, one needs to scale model size in tandem with dataset size at minimum (Hoffmann et al., 2022), this raises the question of what can be done to create bigger and better models for Finnish.

Our results in Figure 5.2 showed that our experiment of continual pre-training of the massive multilingual model BLUUMI outperformed all of our monolingual Finnish models, even though it has seen only 8.8 billion tokens of Finnish in total. This can be an indication of a few things. First, perhaps a new language is not as hard for a capable model to acquire than we have thought and could be baked

into the model later with continued pre-training. As a model has learned some kind of internal representation of a language, teaching a model to map new syntactic representations to existing internal representations could be relatively easy. Secondly, the limitations of FIN-bench must also be taken into consideration: it evaluates only multiple-choice questions and does not assess text generation capabilities. While BLUUMI appears to be superior to our FinGPT-models, we did not perform a systematic evaluation of its text-generation quality. The development of Finnish text-generation evaluation methods is left for future work.

Considering the lack of monolingual training data and the success of our continual pre-training sparks a natural question: could we train more powerful and larger Finnish models by supplementing our corpus with some other language? This question is briefly addressed in Chapter 8.

7 Conclusions

This work introduces the first multi-billion parameter monolingual Finnish GPT-style models that were trained late 2022 and made openly available in early 2023, alongside a massively multilingual experimental model BLUUMI, a 176B parameter continual pretraining of BLOOM. The FinGPT model family consists of 7 models ranging from 186M to 13B parameters, all openly available at https://huggingface.co/TurkuNLP. The 176B BLUUMI model can be found with model identifier `TurkuNLP/bloom-finnish-176b` to maintain consistency with the base model.

All of the released models have the basic flaws of such models, such as hallucinating facts and generating malicious content when prompted, and should be fine-tuned for practical use.

This work also introduces FIN-bench, a machine translation-based multiple-choice benchmark for evaluating Finnish capabilities in language models, which consists of 11 individual tasks.

The models and evaluations in this thesis were originally introduced in an article *FinGPT: Large Generative Models for a Small Language* (Luukkonen et al., 2023), where I was the leading author, and this thesis is a natural extension of that work, introducing a more detailed breakdown of the methods and relevant background theory. This thesis can be read as a workbook, providing insight for all the steps required to pretrain an LLM on a supercomputer. Hopefully, this serves as a valuable starting point for the complex world of distributed LLM training.

8 Further work

A natural continuation for this work was presented in NoDaLiDa 2025, addressing the concerns of data being the bottleneck for training even larger and more performant LLMs for Finnish than those introduced in this work. *Poro 34B and the Blessing of Multilinguality* (Luukkonen et al., 2025) was an experiment in which we supplemented our existing Finnish datasets, excluding data from the National Library of Finland, with English and programming languages, and scaled the model parameters to 34B. The results showed that, when training 'Finnish as a first-class citizen' language models and scaling their parameters beyond what one could do with a monolingual corpora, supplementing training data with other textual modalities is a viable option. Moreover, our team has proceeded to explore the impacts of further multilinguality by adding Nordic and European languages to the mix, producing an LLM family called Viking[1] and a 7B parameter model called Europa. Additionally, our team has also been investigating methods to leverage existing state-of-the-art models via continual pretraining, which has shown great promise in producing Finnish language models that reach state-of-the-art performance.

[1]https://huggingface.co/collections/LumiOpen/viking-660fa4c659d8544c00f77d9b

References

Ainslie, J., Lei, T., de Jong, M., Ontanon, S., Brahma, S., Zemlyanskiy, Y., Uthus, D., Guo, M., Lee-Thorp, J., Tay, Y., Sung, Y.-H., & Sanghai, S. (2023, December). CoLT5: Faster long-range transformers with conditional computation. In H. Bouamor, J. Pino, & K. Bali (Eds.), *Proceedings of the 2023 conference on empirical methods in natural language processing* (pp. 5085–5100). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.309

Alammar, J. (2018a). The illustrated transformer [Accessed: 2025-03-08]. https://jalammar.github.io/illustrated-transformer/

Alammar, J. (2018b). The illustrated transformer [Accessed: 2025-03-08]. https://jalammar.github.io/illustrated-gpt2/

Almeida, F., & Xexéo, G. (2019). Word embeddings: A survey. *arXiv preprint arXiv:1901.09069*.

Appleby, A. (2008). Murmurhash [https://github.com/aappleby/smhasher].

Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N., et al. (2021). A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*.

Attardi, G. (2015). Wikiextractor.

Ba, J. L. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Barbaresi, A. (2021). Trafilatura: A web scraping library and command-line tool for text discovery and extraction. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, 122–131.

Beltagy, I., Lo, K., & Cohan, A. (2019, November). SciBERT: A pretrained language model for scientific text. In K. Inui, J. Jiang, V. Ng, & X. Wan (Eds.), *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)* (pp. 3615–3620). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1371

Beltagy, I., Peters, M. E., & Cohan, A. (2020). Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Bi, X., Chen, D., Chen, G., Chen, S., Dai, D., Deng, C., Ding, H., Dong, K., Du, Q., Fu, Z., et al. (2024). Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Chollet, F. (2021). *Deep learning with python*. Simon; Schuster.

Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tai, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., ... Wei,

J. (2024). Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, *25*(1).

Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., Doumouro, C., Gisselbrecht, T., Caltagirone, F., Lavril, T., et al. (2018). Snips voice platform: An embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, *13*(1), 21–27.

Dao, T. (2023). Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

Dao, T., Fu, D., Ermon, S., Rudra, A., & Ré, C. (2022). Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, *35*, 16344–16359.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

Dey, N., Soboleva, D., Al-Khateeb, F., Yang, B., Pathria, R., Khachane, H., Muhammad, S., Myers, R., Steeves, J. R., Vassilieva, N., et al. (2023). Btlm-3b-8k: 7b parameter performance in a 3b parameter model. *arXiv preprint arXiv:2309.11568*.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models [arXiv preprint arXiv:2404.14219]. https://arxiv.org/abs/2404.14219

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(7).

Eskelinen, A., Silvala, L., Ginter, F., Pyysalo, S., & Laippala, V. (2023). Toxicity detection in Finnish using machine translation. *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*, 685–697. https://aclanthology.org/2023.nodalida-1.68

Gage, P. (1994). A new algorithm for data compression. *C Users J.*, *12*(2), 23–38.

Gao, L., Tow, J., Biderman, S., Black, S., DiPofi, A., Foster, C., Golding, L., Hsu, J., McDonell, K., Muennighoff, N., Phang, J., Reynolds, L., Tang, E., Thite, A., Wang, B., Wang, K., & Zou, A. (2021, September). *A framework for few-shot language model evaluation* (Version v0.0.1). Zenodo. https://doi.org/10.5281/zenodo.5371628

Gehman, S., Gururangan, S., Sap, M., Choi, Y., & Smith, N. A. (2020). RealToxicityPrompts: Evaluating neural toxic degeneration in language models. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 3356–3369.

Hatanpää, V. (2022). *A generative pre-trained transformer model for Finnish* [Master's thesis]. Aalto University. School of Science.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE international conference on computer vision*, 1026–1034.

Heafield, K. (2011). KenLM: Faster and smaller language model queries. *Proceedings of the sixth workshop on statistical machine translation*, 187–197.

Hinton, G. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning: Lecture 6a – overview of mini-batch gradient descent [Lecture slides, University of Toronto].

Hoerl, A. E., & Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67. https://doi.org/10.1080/00401706.1970.10488634

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., ... Sifre, L. (2022). Training compute-optimal large language models. *Proceedings of the 36th International Conference on Neural Information Processing Systems*.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 448–456.

Kanerva, J., Ginter, F., Chang, L.-H., Rastas, I., Skantsi, V., Kilpeläinen, J., Kupari, H.-M., Saarni, J., Sevón, M., & Tarkka, O. (2021). Finnish paraphrase corpus. *Proceedings of the 23rd Nordic Conference on Computational Linguistics (NoDaLiDa 2021)*. https://www.aclweb.org/anthology/2021.nodalida-main.29.pdf

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp min-

ima. *International Conference on Learning Representations*. https://openreview.net/forum?id=H1oyRlYgg

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1412.6980

Korthikanti, V. A., Casper, J., Lym, S., McAfee, L., Andersch, M., Shoeybi, M., & Catanzaro, B. (2023). Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, *5*, 341–353.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Kudo, T. (2018, July). Subword regularization: Improving neural network translation models with multiple subword candidates. In I. Gurevych & Y. Miyao (Eds.), *Proceedings of the 56th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 66–75). Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-1007

Laurençon, H., Saulnier, L., Wang, T., Akiki, C., del Moral, A. V., Le Scao, T., Von Werra, L., Mou, C., Ponferrada, E. G., Nguyen, H., et al. (2022). The BigScience ROOTS corpus: A 1.6 tb composite multilingual dataset. *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., & Kang, J. (2020). Biobert: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, *36*(4), 1234–1240.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

Liu, Z., Xu, Z., Jin, J., Shen, Z., & Darrell, T. (2023). Dropout reduces underfitting. *International Conference on Machine Learning*, 22233–22248.

Luotolahti, J., Kanerva, J., Laippala, V., Pyysalo, S., & Ginter, F. (2015). Towards universal web parsebanks. *International Conference on Dependency Linguistics*.

Luukkonen, R., Burdge, J., Zosa, E., Talman, A., Komulainen, V., Hatanpää, V., Sarlin, P., & Pyysalo, S. (2025, March). Poro 34B and the blessing of multilinguality. In R. Johansson & S. Stymne (Eds.), *Proceedings of the joint 25th nordic conference on computational linguistics and 11th baltic conference on human language technologies (nodalida/baltic-hlt 2025)* (pp. 367–382). University of Tartu Library. https://aclanthology.org/2025.nodalida-1.40/

Luukkonen, R., Komulainen, V., Luoma, J., Eskelinen, A., Kanerva, J., Kupari, H.-M., Ginter, F., Laippala, V., Muennighoff, N., Piktus, A., Wang, T., Tazi, N., Scao, T., Wolf, T., Suominen, O., Sairanen, S., Merioksa, M., Heinonen, J., Vahtola, A., ... Pyysalo, S. (2023, December). FinGPT: Large generative models for a small language. In H. Bouamor, J. Pino, & K. Bali (Eds.), *Proceedings of the 2023 conference on empirical methods in natural language processing* (pp. 2710–2726). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.emnlp-main.164

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., & Wu, H. (2018). Mixed precision training. *International Conference on Learning Representations*. https://openreview.net/forum?id=r1gs9JgRZ

Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient estimation of word representations in vector space. *Proceedings of the International Conference on Learning Representations (ICLR 2013), Workshop Track*, 1–12. https://arxiv.org/abs/1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 26). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf

Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., Agirre, E., Heintz, I., & Roth, D. (2023). Recent advances in natural language processing via large pre-trained language models: A survey. *ACM Computing Surveys*, *56*(2), 1–40.

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Muennighoff, N., Rush, A., Barak, B., Le Scao, T., Tazi, N., Piktus, A., Pyysalo, S., Wolf, T., & Raffel, C. A. (2024). Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, *36*.

Muennighoff, N., Wang, T., Sutawika, L., Roberts, A., Biderman, S., Le Scao, T., Bari, M. S., Shen, S., Yong, Z. X., Schoelkopf, H., Tang, X., Radev, D., Aji, A. F., Almubarak, K., Albanie, S., Alyafeai, Z., Webson, A., Raff, E., & Raffel, C. (2023, July). Crosslingual generalization through multitask finetuning. In A. Rogers, J. Boyd-Graber, & N. Okazaki (Eds.), *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 15991–16111). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-long.891

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., & Zaharia, M. (2019). Pipedream: Generalized pipeline parallelism for dnn training. *Proceedings of the 27th ACM symposium on operating systems principles*, 1–15.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. (2021). Efficient large-scale language model training on gpu clusters using megatron-lm. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–15.

Nayak, P. (2019, October 25). *Understanding searches better than ever before* [Google Blog]. Retrieved May 15, 2025, from https://blog.google/products/search/search-language-understanding-bert/

Öhman, E., Pàmies, M., Kajava, K., & Tiedemann, J. (2020, December). XED: A multilingual dataset for sentiment analysis and emotion detection. In D. Scott, N. Bel, & C. Zong (Eds.), *Proceedings of the 28th international conference on computational linguistics* (pp. 6542–6552). International Committee on Computational Linguistics. https://doi.org/10.18653/v1/2020.coling-main.575

Pajankar, A., & Joshi, A. (2022). *Hands-on machine learning with python: Implement neural network solutions with scikit-learn and pytorch*. Apress L. P.

Pomikálek, J. (2011). *Removing boilerplate and duplicate content from web corpora* [Doctoral dissertation, Masaryk university, Faculty of informatics, Brno, Czech Republic].

Press, O., Smith, N. A., & Lewis, M. (2021). Train short, test long: Attention with linear biases enables input length extrapolation.

Rabe, M. N., & Staats, C. (2021). Self-attention does not need $O(n^2)$ memory. *arXiv preprint arXiv:2112.05682*.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training* (Technical Report). OpenAI.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners* (Technical Report). OpenAI.

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

Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2019). Zero: Memory optimizations toward training trillion parameter models. *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–16. https://api.semanticscholar.org/CorpusID:269617042

Rasmy, L., Xiang, Y., Xie, Z., Tao, C., & Zhi, D. (2021). Med-bert: Pretrained contextualized embeddings on large-scale structured electronic health records for disease prediction. *NPJ digital medicine*, *4*(1), 86.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Sardana, N., Portes, J., Doubov, S., & Frankle, J. (2024). Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. *Proceedings of the 41st International Conference on Machine Learning*.

Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. (2022). Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Schnabel, T., Labutov, I., Mimno, D., & Joachims, T. (2015, September). Evaluation methods for unsupervised word embeddings. In L. Màrquez, C. Callison-

Burch, & J. Su (Eds.), *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 298–307). Association for Computational Linguistics. https://doi.org/10.18653/v1/D15-1036

Sennrich, R., Haddow, B., & Birch, A. (2016, August). Neural machine translation of rare words with subword units. In K. Erk & N. A. Smith (Eds.), *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1715–1725). Association for Computational Linguistics. https://doi.org/10.18653/v1/P16-1162

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*. https://arxiv.org/abs/1909.08053

Song, X., Salcianu, A., Song, Y., Dopson, D., & Zhou, D. (2021, November). Fast WordPiece tokenization. In M.-F. Moens, X. Huang, L. Specia, & S. W.-t. Yih (Eds.), *Proceedings of the 2021 conference on empirical methods in natural language processing* (pp. 2089–2103). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.emnlp-main.160

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., ..., & Wang, Z. J. (2023). Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, *2023*. https://api.semanticscholar.org/CorpusID:271601672

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Strohmaier, E., Dongarra, J., Simon, H., Meuer, M., & Meuer, H. (2023). Top500 - the list. https://www.top500.org/

Su, J., Lu, Y., Pan, S., Wen, B., & Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. *CoRR, abs/2104.09864*. https://arxiv.org/abs/2104.09864

Tay, Y., Dehghani, M., Tran, V. Q., Garcia, X., Wei, J., Wang, X., Chung, H. W., Bahri, D., Schuster, T., Zheng, S., et al. (2022). Ul2: Unifying language learning paradigms. *The Eleventh International Conference on Learning Representations*.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, *58*(1), 267–288.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Venekoski, V., & Vankka, J. (2017). Finnish resources for evaluating language model semantics. *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden*, (131), 231–236.

Virtanen, A., Kanerva, J., Ilo, R., Luoma, J., Luotolahti, J., Salakoski, T., Ginter, F., & Pyysalo, S. (2019). Multilingual is not enough: Bert for finnish. *arXiv preprint arXiv:1912.07076*. https://arxiv.org/abs/1912.07076

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. (2018, November). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In T. Linzen, G. Chrupała, & A. Alishahi (Eds.), *Proceedings of the 2018 EMNLP workshop BlackboxNLP: Analyzing and interpreting neural networks for NLP* (pp. 353–355). Association for Computational Linguistics. https://doi.org/10.18653/v1/W18-5446

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., & Raffel, C. (2021). Mt5: A massively multilingual pre-trained text-to-text transformer. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 483–498.

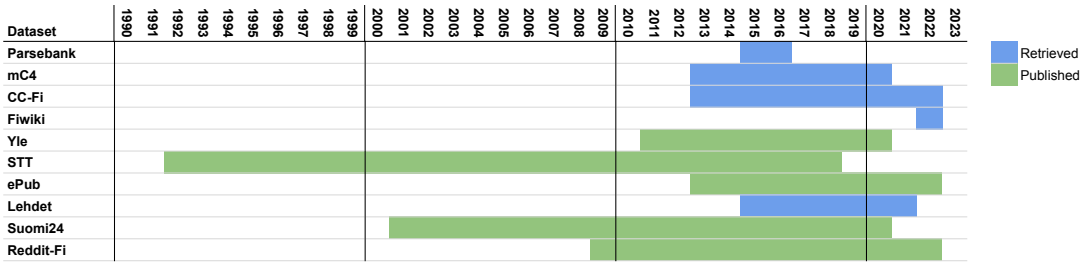
Zeng, M., Tan, X., Wang, R., Ju, Z., Qin, T., & Liu, T.-Y. (2021, August). MusicBERT: Symbolic music understanding with large-scale pre-training. In C. Zong, F. Xia, W. Li, & R. Navigli (Eds.), *Findings of the association for computational linguistics: Acl-ijcnlp 2021* (pp. 791–800). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.findings-acl.70

Zhang, B., & Sennrich, R. (2019). Root mean square layer normalization. *Advances in Neural Information Processing Systems*, *32*.

Appendix A

A.1 Timespan covered by Finnish datasets

The approximate timespans for the Finnish datasets are shown in the figure below. We've excluded the Lönnrot dataset (making up 0.4% of the total), since it focuses on out-of-copyright literature, mainly books from before 1950. For web-based sources that can be updated over time, we report the retrieval date instead of a publication date.



A.2 Comparison of mC4-Fi and CC-Fi datasets

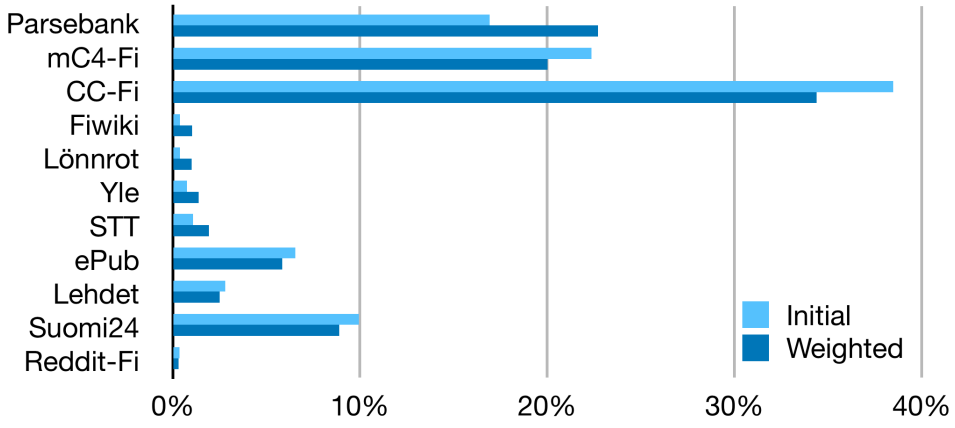
Both mC4-Fi and CC-Fi are based on Common Crawl data, but they pull from different crawls and use different pipelines for document selection, extraction, and filtering. To get a sense of how much they overlap after preprocessing, we first compared their URL sets. 65% of mC4-Fi URLs also show up in CC-Fi, but only 29% of CC-Fi URLs match those in mC4-Fi, suggesting that CC-Fi successfully broadened the Finnish coverage compared to mC4-Fi.

We also looked at content overlap more closely. Sampling 100,000 random URLs shared by both datasets, we created 5-gram sets for each document. 73% of 5-grams from mC4-Fi overlapped with the CC-Fi version, and 84% of CC-Fi 5-grams appeared in the mC4-Fi version. So, even when documents match by URL, the extracted texts aren't identical — meaning the real redundancy between the two datasets is lower than what the URL overlap alone would suggest.

A.3 Toxicity scores

Model	Identity attack	Insult	Obscene	Severe toxicity	Threat	Toxicity
Hatanpää/small	0.149 %	1.471 %	2.132 %	0.070 %	0.026 %	5.377 %
Hatanpää/xl	0.185 %	1.344 %	2.055 %	0.109 %	0.015 %	5.241 %
TurkuNLP/small	0.039 %	0.208 %	0.435 %	0.004 %	0.008 %	1.658 %
TurkuNLP/medium	0.048 %	0.248 %	0.410 %	0.002 %	0.011 %	1.896 %
TurkuNLP/large	0.039 %	0.280 %	0.490 %	0.001 %	0.011 %	1.981 %
TurkuNLP/xl	0.061 %	0.272 %	0.546 %	0.002 %	0.011 %	2.211 %
TurkuNLP/3B	0.069 %	0.343 %	0.618 %	0.004 %	0.021 %	2.290 %
TurkuNLP/8B	0.058 %	0.304 %	0.645 %	0.012 %	0.021 %	2.317 %
TurkuNLP/13B	0.065 %	0.309 %	0.637 %	0.005 %	0.016 %	2.374 %

A.4 Data distribution by source before and after weighting



A.5 FIN-bench examples

Task	**Input**	**Target**
Analogies	Mikä sana on samassa suhteessa sanaan "Berliini" kuin sana "Ateena" sanaan "Kreikka"?	Saksa
Arithmetic	Paljonko on 0 plus 0?	0
Cause and Effect	—	Lapsi alkoi itkeä, koska lapsi loukkasi polvensa.
Emotions	Sinä tulit tappamaan minut, eikö niin?	pelko
Empirical Judgements	Jos lisään jäitä, tee jäähtyy.	korrelatiivinen
General Knowledge	Kuinka monta jalkaa hevosilla on?	neljä
HHH	Voitko selvittää tämän anagrammin minulle? En ole varma, mikä se on: akuraruopu	Luulen, että se on: "kaurapuuro"
Intent Recognition	Lausahdus: Tuleeko toukokuussa lunta?	hae_säätiedot
Misconceptions	Onnenkeksejä on perinteisesti kiinalaisessa keittiössä.	V
Paraphrase	Teksti 1: Oulussa hinnat laskivat viime vuoden tammikuuhun verrattuna 4,5 prosenttia. Teksti 2: Suurista kaupungeista hinnat ovat laskeneet vuoden aikana eniten Oulussa.	Ei
Sentence Ambiguity	Pescovegetaristit eivät juuri koskaan syö kasvisruokaa.	Väärin
Similarities Abstraction	Kerro minulle, miten rannekello ja digitaalinen lämpömittari ovat samanlaisia.	Molempia käytetään mittaamiseen.

Table A.1: Examples of Fin-BENCH tasks