Bachelor Thesis

Neural scaling laws: the impact of large language model ensembles on their performance

Arthur Allebrandt Werlang

March 31, 2025

Reviewer:
Prof. Dr. Emmanuel Müller
Simon Klüttermann

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl 9 - Data Science and Data Engineering
https://ls9-www.cs.tu-dortmund.de

Abstract

This thesis explores the effectiveness of ensembling large language models in relation to the scaling laws defined by Kaplan et al. By training 13 models with varying configurations while not allowing the dataset and compute budget to limit our results, we analyzed how ensembling affects performance when using the *average*, *minimum*, and *maximum* functions as aggregators. Our findings show that while ensembles consistently outperform their worst-performing base models, they do not scale as efficiently as single large models and often fail to surpass their best base models. These results suggest that ensembling, while useful for marginal improvements, is not a substitute for directly training larger models but could serve as an alternative tool to improve performance when multiple models are already available.

Contents

Abstract i

1 Introduction 1
- 1.1 Motivation and Background . . . 1
- 1.2 Thesis Structure . . . 1

2 Large Language Models 3
- 2.1 Background . . . 4
  - 2.1.1 Loss and gradient descent . . . 4
  - 2.1.2 Tokenization . . . 6
  - 2.1.3 Transformers and attention . . . 7
  - 2.1.4 Generative pre-trained transformer . . . 9
- 2.2 Scaling Laws . . . 11
  - 2.2.1 A summary of scaling laws . . . 11

3 Ensembles 14
- 3.1 Combining results . . . 14
  - 3.1.1 Aggregating predictions . . . 14
  - 3.1.2 Bagging . . . 14
  - 3.1.3 Boosting . . . 15

4 Methodology and Results 16
- 4.1 Results . . . 18
  - 4.1.1 Training time . . . 18
  - 4.1.2 Replicating parameter scaling laws . . . 21
  - 4.1.3 Ensembling . . . 22
  - 4.1.4 Ensembling similar models . . . 25

5 Conclusion 28

Bibliography 31

1 Introduction

1.1 Motivation and Background

In the realm of artificial intelligence and machine learning there are still a lot of possibilities and insights to be explored. Having that as a frame of mind, finding new ways to improve training, performance, and efficiency is of utmost importance to further develop the field. In their paper [9] Kaplan et al found that we can see multiple relations between a neural language model's performance (log-likelihood loss, for example) and their features - in this thesis, the amount of parameters is the one which we lay our focus upon.

They observed a consistent scaling between that loss and the non-embedding parameter count, which is a very interesting find, that raised the question of how ensembles would fit in these scaling laws defined by the authors. While these laws provide us with a base to work and predict performance when training new models, they limit and constrain what should be possible - overcoming these laws could provide us with a new breakthrough in the field. Thus it is of importance to test these boundaries to see if we can break them, and push our development of large language models forward. This is the main goal of this thesis, with a focus on ensemble experiments, trying to find their place in the laws defined by other researchers, while using the laws themselves as the benchmark to be beat.

In their study of correlated representations in contrastive learning, Klüttermann et al [12] observed that while using an ensemble of models instead of a single model better results could be achieved, having less bias, as non-correlated representations of one model did not match the ones from other models. They managed to get significantly better results while keeping the amount of parameters at a similar level. As a result, this provided us with the desire to analyze if similar effects could be reproduced in large language models, significantly increasing the performance without changing radically the amount of parameters and without extra training, and thus analyzing how that would interact with the scaling laws of Kaplan et al.

To achieve this, 13 models were trained - their losses, parameters, and training time were analyzed. From this base, we experimented with ensembling, its effects, and outcomes. Different methods of aggregating predictions were tested, and the relations to the scaling laws were measured and examined in the search for a performance improvement that would pull apart from the laws devised by Kaplan et al.

1.2 Thesis Structure

This thesis is divided into 6 chapters.

- The first chapter deals directly with large language models. It presents a background into their history, and then it moves into the technical background of LLMs, from loss func-

tions, and gradient descent, to the transformer architecture and generative pre-trained transformers.

- The second chapter presents a short introduction to ensembles, how they work, why they work, and the different methods one can use to achieve a desired end result using ensembling.
- The third chapter consists of the methodology of this thesis. What was done, which experiments were conducted, what tools were used, and it presents all results from the experiments conducted in this thesis. It explains the experiments, and how they were conducted in more detail and presents hypotheses and conclusions taken from those outcomes.
- The last chapter summarizes the findings of this thesis, concludes it and presents a discussion about what can be further researched in the future, complementing this study.

2 Large Language Models

Language is at the roots of human development. Working together with our peers is what makes science possible the way it is and behind all that lies a powerful tool, often overlooked, as it is so common and so present in our lives: language. From the moment we are born, we are exposed to it, and human lives are molded by it. Computers, on the other hand, lack the capacity to intrinsically comprehend and communicate in human language, and to solve that language modeling [22] was devised. *Natural language processing (NLP)* emerges as the branch dedicated to the creation of machines capable of reading, writing, understanding human language - and much more through their emergent abilities [24].

In the beginning, researchers developed *statistical language models (SLMs).* These mere mathematical models would analyze properties of natural language from a probabilistic statistical perspective [22]. Take for example the following sentence: "I come from Brazil". $P(\omega_i)$ represents the probability of a word ω_i, with i being the i-th word of a sentence. Now for the likelihood of the whole sentence, we have:

$$P(\omega_1, \omega_2, \omega_3, \omega_4) = P(I, come, from, Brazil)$$

To calculate that, we need to employ conditional probability until we get the probabilities for this specific sentence. To get the probabilities for each word, maximum likelihood estimation would be used, by taking into account the frequencies of each word as the i-th member of the sequence, by also analyzing the rest of the latter. The problem with this approach would be the huge amount of possible sequences, which posed storage limitations and thus a reduction in the model's accuracy [22].

To solve that, researchers started leveraging neural networks to do this task. Firstly, they needed to tokenize and vectorize inputs, so that the algorithms could interpret them. We explore this topic deeper in the following subsections. These new models, called *neural language models* could thus take longer sequences into account, and generate much better results. They still worked similarly to SLMs, but were much more complex and effective. Pre-trained language models would also soon become the norm, as a type of neural language model, as they allowed the use of even bigger datasets in a semi-supervised learning environment.

From that, we got the first large language models (LLMs), which are more present in the everyday lives of all of us than ever. With the rise in popularity of models such as *ChatGPT*, *LLaMMA*, *DeepSeek*, and others, the bigger becomes the importance of understanding the underlying mechanisms behind these technological advancements and their origins, which we will explore in this chapter briefly.

LLMs are advanced artificial intelligence systems trained on vast amounts of data and parameters to understand and generate human-like language. Using deep learning techniques, particularly the transformer architecture [20], LLMs analyze patterns in a text to predict and

generate coherent responses, answer questions, summarize information, and even create content. These models, process input by breaking it into tokens [23][13] and leveraging thousands to trillions of parameters to determine the most contextually relevant output. These models are also able to develop emergent abilities, which are only visible after a certain size and scale [24], and allow them to be incredible generalists for many different applications.

Their applications range from chatbots - such as ChatGPT from OpenAI - to virtual assistants for code generation and content creation, making them powerful tools for various industries. The cutting-edge models available can already handle images, being able to generate them on the spot based on prompts, to describing them back in detail for people who cannot see. Their applications are extremely diverse, as language permeates everything in our lives.

2.1 Background

2.1.1 Loss and gradient descent

One of the pillars of machine learning, is how we can evaluate whether results are being meaningfully generated, our samples correctly classified, or represent well real-life expectations. To do that, we use loss functions. Also called cost functions or error functions, they calculate the cost of a specific decision [6]. A simplistic way of explaining them for machine learning would be a functions which give the difference between expected and actual values. As a result, when expected and predicted values are very similar according to our loss functions, we get a very small error, and the opposite for big discrepancies. If both match, we have an error of 0. For such an end, one can use many different methods, for example, the squared error loss [6]:

$$L(Y, f(X)) = (Y - f(X))^2$$

Where L is the loss function, Y the actual values, and $f(X)$ our model prediction. We square the difference between them so that our error stays always positive.

Another very robust loss function is *cross-entropy loss*. Cross-entropy loss is characterized by the use of the $\log$ function, and by resulting in values between 1 (worst case) and 0 (best case). In this thesis, we use cross-entropy loss directly to measure the performance of our models and ensembles.

$$L(Y, f(X)) = -\left[Y \log(f(X)) + (1 - Y) \log(1 - f(X))\right]$$

This loss function is the one used for the training of many language models, for it aligns well with probability-based predictions. Many models use for example the *softmax* function in their predictions, with whom the loss function interacts very well by penalizing bad decisions and reinforcing correct ones, making it a very robust alternative in the realm of neural language models.

Also very important in this topic, is *gradient descent*. Gradient descent helps us directly to minimize the loss of our models, by mathematical optimization. It works only on defined and differentiable functions, as we use differentiation directly to calculate the direction of our descent/ascent. The main idea of it is to take steps towards the gradient of the function we are analysing, and the size of these steps are called *learning rate* [8].

The learning rate itself defines how big our jumps within the gradients are going to be. Smaller learning rates result in smaller steps - that can be beneficial, as converging much more precisely to a local minimum becomes easier. On the other hand, with such small steps, we might get stuck in suboptimal local minima, or take too long to reach proper results. Too high learning rates can also cause problems: it can make us jump over local minima (overshoot) and thus never find the desired results. As a result, in most cases, the learning rate is varied during training, either by a fixed schedule or by using an adaptive learning rate. This makes sure we get "the best of both worlds", having the precision of low learning rates, with the practicality as speed of high ones combined.

So for a loss function F, with an $F(x)$ in the neighborhood of a point z, the function F will decrease the fastest in the direction of the negative gradient [8]. So by calculating the negative gradient of the function $-\nabla F(z_1)$ and multiplying it by our learning rate γ and summing it to the previous position, we get

$$z_{n+1} = z_n - \gamma \nabla F(z_n)$$

and if our learning rate is reasonable, we should get that $F(z_n) \geq F(z_{n+1})$.

Inside this topic, we can define different types of gradient algorithms. One of these types is the *first-order gradient optimization algorithms* or *steepest descent method*. These algorithms are characterized, as the name implies, by the use of the first-order gradient, as we saw in the previous examples. They are not as powerful in some sense, compared to other possible methods, such as the Newton Method [8], but because of their simplicity, small memory footprint, and efficiency, they are used by most large-scale machine learning algorithms nowadays.

An example of these algorithms is the *batch gradient descent*, which at each iteration uses the entire training dataset to calculate the next gradient vector [8].

$$z_{n+1} = z_n - \gamma \nabla F(z_n; (X, Y))$$

It is a powerful solution that guarantees convergence in convex datasets, but that can be extremely slow in the case of large datasets, that sometimes cannot even fit into memory.

This led to advancements and the development of other algorithms that took advantage of different techniques. One of these algorithms is called Adam [8]. The Adam optimizer [11] is a widely used first-order gradient-based, used as the algorithm by Kaplan et al in their study of scaling laws [9]. It maintains two moving averages: one for past gradients, denoted as m, and another for the squared gradients, represented as v. These help adjust the learning rate dynamically for each parameter.

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) g(\theta_t)$$

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2) g^2(\theta_t)$$

Here, m_t represents the exponentially weighted average of previous gradients with a decay factor β_1, while v_t tracks the average of squared gradients with a decay factor β_2. Typically, β_1 and β_2 are set close to 1, which may lead to slow updates at the beginning of training due to initial bias. To mitigate this issue, bias-corrected estimates are introduced:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Once the bias-corrected estimates are obtained, they are used to update the model parameters as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

The default hyperparameter values for Adam are $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$. Adam has gained widespread popularity in deep learning due to its efficiency, stability, and adaptability, requiring minimal manual tuning of learning rates compared to other optimization methods, whilst also solving the problems present in batch gradient descent, as we do not need to analyze the whole dataset to get gradient results.

In conclusion, gradient descent, loss functions, and learning rate are key concepts in machine learning in general, that also closely apply to the world of language models. Loss or error functions are the benchmarks for models whether their performance is satisfactory and reaching the desired outputs. Without it training is not possible, and thus machine learning itself as well. Gradient descent, strongly linked to the former, is what allows us to find the best way to reduce our loss and get to the desired results, by using the gradients of functions, most commonly the first-order gradient. Learning rate comes then into play, as we need to carefully find the best possible step sizes in many situations, so that we get both effective and fast progress, as precision in the search for local minima.

2.1.2 Tokenization

Tokenization is an important step in the pre-processing done for large language models [23]. As humans, we usually take the meaning of words for granted, but for algorithms, we need to find some new way of giving and storing meaning in ways that we can easily reproduce and retrieve later. Tokenization is the process of decomposing information and defining tokens, these basic units of knowledge which can be used by machine learning algorithms to learn and predict results. At first, this meant splitting content into sentences or words, due to linguistic motivations. Nowadays the most effective methods are those that split even further, with words being split into smaller subwords which are than used as tokens [13]. These tokens are thus non-linguistically motivated units and do not carry the same meaning as old tokenization methods would use (per word/sentence tokenization). One of the modern subword tokenization algorithm is *Byte-Pair-Encoding (BPE)* [13][5].

Byte Pair Encoding is a tokenization technique that iteratively merges frequently occurring pairs of adjacent symbols into a single token. The process begins with a basic vocabulary consisting of individual characters or bytes. At each step, the most common adjacent symbol pair is identified and replaced with a new token representing that pair. This merging process continues until a predefined vocabulary size is reached.

BPE is widely used in natural language processing tasks, particularly in subword tokenization for neural network-based models, as it helps efficiently handle rare words and reduces the overall vocabulary size while preserving word structures.

To further read about tokenization, see [13].

2.1.3 Transformers and attention

On the paper "Attention is All You Need" [20], Vaswani et al introduced the world the concept of the *transformer*. Originally, transformers were focused on translation tasks, but with time, they started being used in multiple facets of machine learning, from natural language processing to computer vision. The transformer was a new architecture focused directly in the attention mechanism and had much better performance than past strategies, such as the use of recurrent neural networks. The type of attention in transformers is called self-attention.

This new attention mechanism allowed an edge over other architectures, as it made it possible for models to have better context and long-term memory. Different models, using RNNs for example, have trouble with sentences like "The quick brown fox jumped over the lazy dog, that just stared at it". In this sentence *"it"* refers to the fox, but a normal RNN would struggle to link this, and produce suboptimal results, as its lack of an attention mechanism limits the memory of the model. Transformers mitigate this in the way that they use multiple heads of attention, that give different tokens different "attention", or rather scale their vectors so that they remain important in the context we are in at the moment.

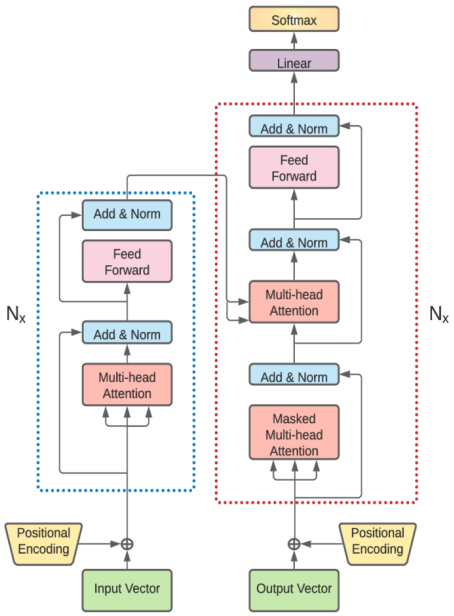


Figure 2.1: Basic transformer architecture [1].

The specific attention function used in transformers is called *scaled dot-product attention*.

The input of this function is the queries and keys, with dimension d_k, and the values with dimension d_v. With this, we calculate the dot product of queries and keys, divide it by $\sqrt{d_k}$, and use a softmax function to get their respective weights.

In the actual use of the transformers, we use vectors for each of the inputs, being Q the queries, K the keys, and V the values.

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

Now to further increase the performance, we use multi-head attention. Instead of using a single attention function, the authors in [20] found it beneficial "to linearly project the queries, keys, and values h times with different, learned linear projections do d_k, d_k and d_v dimensions, respectively". These allow us to run the attention function in parallel, concatenate the results, and project them once again, generating the final values. Having multi-head attention makes it possible to consider information from different subspaces at different positions.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, \quad W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}} [20].$$

This is then used by transformers in three different ways [20]:

- Encoder-decoder attention, where the queries come from the previous decoder layers, while the keys and values from the output of the encoder;
- Encoder containing self-attention layers, where all keys, values, and queries come from the output of the previous encoder layer.
- Decoder with self-attention layers, that allow each position in it to attend all positions up to and including that position.

Layers in the encoder and decoders also contain fully connected feed-forward networks, which are applied to each position separately and identically [20]. Another important feature of transformers is their embeddings. Using the learned embeddings, transformers convert input and output token to vectors, which using the *softmax* function, convert the decoder outputs to probabilities of new tokens. Also because there is no recurrence and no convolution, we need to insert the position of tokens into embeddings.

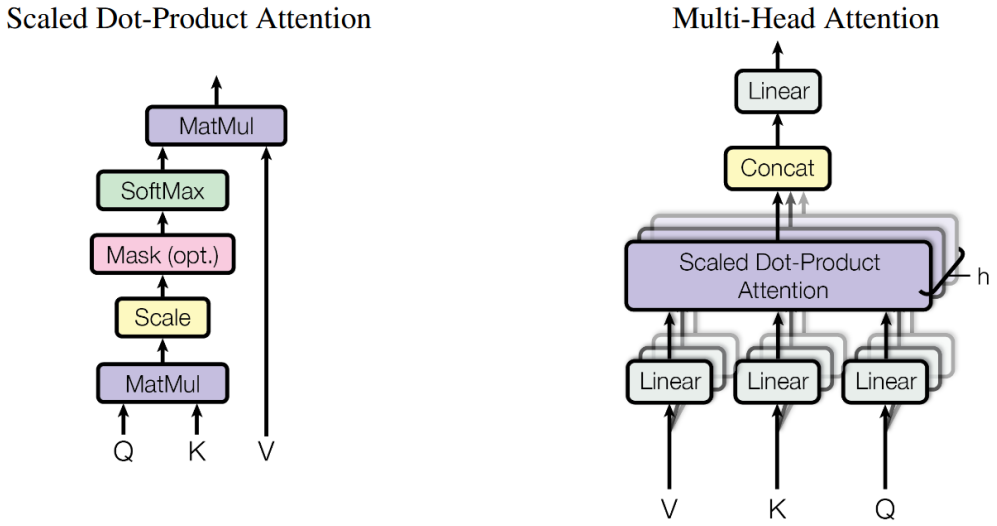


Figure 2.2: Visual representation of both scaled dot-product attention (left) and multi-head attention (right) [20].

2.1.4 Generative pre-trained transformer

One of the most famous uses of transformers is the *generative pre-trained transformer*, usually just called *GPT*. GPTs are the basic building blocks of modern applications such as ChatGPT, which is propelled by the use of generative pre-training [15]. This pre-training deals directly with one of the difficulties of training large language models: the dataset to train on, which must often be manually labeled by researchers. Pre-training is rooted in transfer learning [21], and thus allows the model to train in large, unlabeled, raw text datasets, which facilitates the training of big models that perform well in different scenarios. It was first developed as a form of semi-supervised machine learning, so that first the models would be trained on unlabeled datasets, and then would be tested using labeled datasets, bettering the results, whilst keeping the effort needed to label and categorize manageable. When done in encoder-decoder transformers it can lead to a unified model that can both understand and generate in the context of language. Before that, most studies would suggest the use of specific models for specific tasks, with high specialization.

GPT-1 was the first of OpenAI's transformer based models [15]. Up until this point, the best models one could find in the neural language procession landscape were all supervised models. The capability of training a model unsupervised with extra fine-tuning afterward provided the breakthrough needed in the field. It was trained on *BookCorpus*, an unpublished books database, and had a size of 117M parameters [15].

After the development of the first generative pre-trained transformers, a very important checkpoint was the introduction of *GPT-2* [16]. GPT-2 was a model developed by OpenAI and

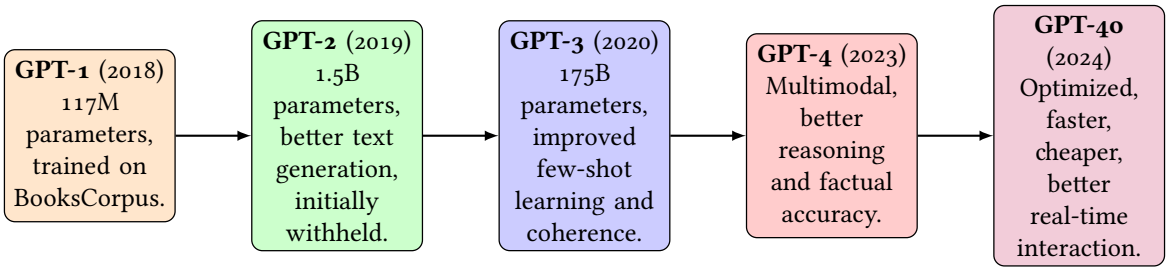


Figure 2.3: Timeline of GPT models

released in 2019. It was a direct improvement upon GPT-1, the first generation of the GPT models, by having over an order of magnitude more parameters. It could zero-shot - meaning perform tasks not in the training data, without demonstrations - and outperform most of the state-of-the-art models back in its release year. Models can also ”few-shot”, where they receive as many demonstrations as their context window allows, or one-shot, meaning they have one example in their input showing how to perform the task [2].

At the core of GPT-2 lies language modeling [16]. Because language is mostly sequential, we can use a model to predict the next term in this sequence - the next token, the next word. By doing that, we have results that can correspond to our expectations. Most times, however, there are too many possible answers for a single problem or many ways of defining a single task. To calculate that, we need not only a probability for each situation, this probability must depend on input and the task at hand. Transformers [20] helped solve this directly, by using self-attention, which basically allows us to encapsulate this task into the meaning of an input. This allowed GPT-2 to zero-shot to state-of-the-art performance in many language modeling datasets, without direct supervision.

After the development of GPT-2, came the bigger and better GPT-3 [2]. Both models are very similar in architecture, both being decoder only and generative pre-trained transformers. The main difference lies in GPT-3’s size: it has circa 175 billion parameters, resulting in a much larger model than its predecessor. It could also few-shot and one-shot to state-of-the-art in some tasks, and would also perform very promisingly in zero-shot scenarios. Trained on 5 very large datasets, *Common Crawl* being the biggest, generated great results, that could unfortunately sometimes also be toxic - *Common Crawl* contained unfiltered data from all over the internet, and thus also fowl language and toxic behavior. Many measures were put in place by OpenAI to limit the toxicity, which worked and reduced it to much lower levels.

The latest iteration of generative pre-trained transformer models is GPT-4 [3]. The model can perform at human level in many different tasks but still suffers from some of the problems of older generations, such as hallucinations (making up information). This was one of the aspects the team of developers tried to improve upon, testing the model with real-world exams, and grading the results. GPT-4 was able, for example, to pass the bar exam with a performance in the top 10% of test takers [3].

GPT-4 also is a multimodal model, meaning it can also take images as input, interpret them, and complete tasks based on that. This represents a new step toward general-use artificial in-

telligence, with powerful capabilities that can permeate our entire lives - from chatbots that can answer simple questions, translate from different languages in a natural manner, to seeing and describing images for people with impaired vision. Soon after came also *GPT-4o*, an optimized version of GPT-4, working more efficiently whilst delivering even more. GPT-4o is capable of processing and generating text, images, and even audio [14], making it incredibly versatile.

The evolution of GPT models from GPT-1 to GPT-4o showcases the rapid advancements in language model capabilities. Each new iteration has brought significant improvements, from the introduction of unsupervised pre-training in GPT-1 to the zero-shot, few-shot, and one-shot learning capabilities of GPT-2 and GPT-3. With GPT-4, we saw a leap in reasoning abilities and multimodal inputs, further expanding the model's practical applications.

Now, with GPT-4o, efficiency and versatility have reached new heights, integrating text, image, and even audio processing into a single model. These advancements highlight not just the growth of AI-powered language models, but also their increasing role in real-world applications. While challenges like hallucinations and bias still exist, continuous research and improvements push the boundaries of what AI can achieve, moving us closer to more general and reliable artificial intelligence systems.

2.2 Scaling Laws

On the topic of training large language models, efficiency is of utmost importance. The best performance possible, with the least amount of compute power, training, and model size is a common goal throughout all machine learning models, non-excluding neural language ones. Specifically on this topic, Kaplan et al performed extensive research [9], and defined scaling laws of neural language models. Using the transformer architecture as the base, they evaluated the impact of different factors - such as parameter count, compute time and dataset size - on the loss of a model.

Whilst the scaling laws are great ways of predicting performance, they inherently limit what can and cannot be achieved. This opens many possibilities for research when looking for ways to break free from these constraints and achieve even better performance - one of them being the ensembling of LLMs.

2.2.1 A summary of scaling laws

From their research [9], Kaplan et al made several observations that resulted in the formulation of their scaling laws:

- Scale is more relevant to performance than model shape. Depth and width of the model's architecture are not as important - what matters most are model size, dataset size, and amount of compute, where more of all of them equals better performance.

- We see smooth power laws. Performance scales directly with dataset size, parameter count (non-embedding) and compute power, when not limited by one of the others.

- If we scale parameter count but not dataset size, and vice versa, overfitting happens. We do see better performance, but strongly diminishing returns. Scaling both in proportion guarantees predictably better results.
- Training curves follow power laws that are roughly independent on model size. From the early results of the training curve, we can extrapolate its development. This allows us to conduct experiments using a much smaller scale than the original research paper, facilitating training overall.
- When transferring models to different datasets, there is a strong correlation between training/validation loss, and the loss in the new datasets. The results are strongly correlated, there is only a mostly constant offset in the loss.
- Smaller models are not more sample-efficient. Larger models reach the same sample efficiency with fewer optimization steps.
- When we have a limited compute budget, the best performance is achieved by training very large models and stopping shy of convergence.
- The optimal batch size is roughly a power of only the loss. It can be determined by measuring the gradient noise scale.

Thus we conclude the loss of a transformer trained autoregressively can be predicted using a power-law when limited only by one of the factors (dataset size, non-embedding parameter count and compute budget). Considering N as the number of non-embedding parameters, C as the compute budget, and D as the dataset size, the following power-laws are thus applicable [9]:

1. When the number of parameters is limited, on a large enough dataset, trained to convergence (no limitations on compute):

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}, \quad \alpha_N \sim 0.076, \quad N_c \sim 8.8 \times 10^{13} \text{ (non-embedding parameters)}$$

2. For large models trained with a limited dataset with early stopping:

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D}, \quad \alpha_D \sim 0.095, \quad D_c \sim 5.4 \times 10^{13} \text{ (tokens)}$$

3. With a limited amount of compute budget, big enough dataset and optimally sized model and small batch size:

$$L(C_{\min}) = \left(\frac{C_c^{\min}}{C_{\min}}\right)^{\alpha_C^{\min}}, \quad \alpha_C^{\min} \sim 0.050, \quad C_c^{\min} \sim 3.1 \times 10^8 \text{ (PF-days)}$$

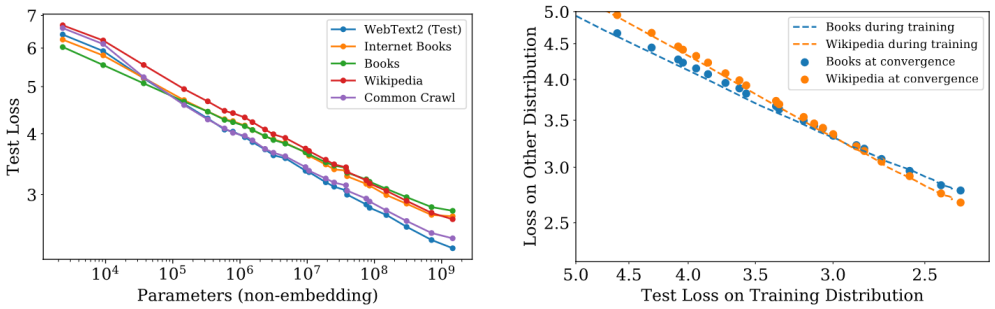


Figure 2.4: Relation between loss and parameter count. The Loss goes down on the test split on all tested datasets - and this transfers also to other distributions. [9]

These consistent scaling relations allow us to even predict loss results from the information we have, bringing the study not only observational importance but also prediction capabilities that can be used in the training of all sorts of language models. Examples are finding the compute scaling, magnitude of overfitting, early stopping step, and data requirements when training large language models [9].

In short, the scaling laws found by Kaplan et al give us a clear way to understand and improve the training of large language models. They show how parameter count, dataset size, and compute budget interact, helping researchers make better choices about how to use resources efficiently. But they constrain and limit what should be achievable, and taking them as absolute laws of nature can be detrimental to the future development of large language models. The possibility of breaking these laws and finding even better bounds for performance by using ensembling is the direct goal of this thesis, as also analyzing and finding new paths for future research on the topic.

3 Ensembles

Ensembles of machine learning models are a powerful tool in the search for efficiency and performance in the artificial intelligence world. They are a conglomerate of smaller, base learners or base predictors that may or may not work on their own [19]. They can be formed of multiple types of models - including large language models - and many different methods can be used to agglomerate their results cohesively. Examples could go from simple such as using mean, minimum, and maximum values, to boosting, bagging, and other more complex solutions which will be explored in the following subsections.

The main power of ensembles comes from the diversity in their models: different models learn different aspects of datasets, and when put together, their different opinions can be used to find some common ground, which may yield better results without training new models [19]. For example, in their research on the topic of contrastive learning, Klüttermann et al observed that ensembling multiple models resulted in a better performance than having single bigger models. They present that the outcome may be such, as the hidden layers of each model tend to learn different features in the datasets, and then when combined, they form a cohesive and more robust prediction model [12].

3.1 Combining results

3.1.1 Aggregating predictions

Aggregating methods are usually the simplest to implement. A simple example would be averaging the results between the different base predictors for regression tasks, which is one of the methods used in the experiments presented in this thesis. In this specific method, the average of all predictions of all base learners is calculated, finding some sort of middle ground between all predictions, that might be closer to the values we are looking for.

For classification, another possibility is the use of majority voting. The ensemble takes the classification results from each base learner as a vote, and the one with the most votes is considered the end result. Some other forms of aggregating can be done by using machine learning itself to create models that pick predictions or parts of predictions from the different base learners to in the end reach a single conclusion, which is called stacking [19].

3.1.2 Bagging

Bagging is a method characterized by the fact that many weak learners can be trained in parallel and independently. One of the famous bagging methods is *Random Subspaces* [7]. When using random subspaces, each base learner (usually a decision tree) has access to the whole dataset but only analyses a subset of the dimensions. As a result, we end up having multiple models

that can be a very strong fit for their dimensions, as each tree is given as many split nodes as needed to 100% accuracy in the training. However, these models are still considered weak predictors when it comes to the whole dataset. To get a final result, we can take all predictions from all weak learners and through majority voting categorize it. In the case of regression, using the average of all models is also a common solution.

3.1.3 Boosting

A different way of approaching the problem, is by using *boosting*. Boosting is a way of getting better results, in which we add new base learners to an ensemble sequentially, in a way that new models pay closer attention to the errors committed by the previous iteration [19]. This can of course be done in multiple ways, which results in multiple algorithms.

For example, the first boosting algorithm [17] [19] was very simple, using only 3 base learners. The first model would train on the whole dataset and make its predictions. The second model would then only train using a subset of the dataset where half the data points were the ones the first base learner predicted falsely. The third one would only train on the data points where both models disagreed if there were any. The end result would be chosen based on a majority vote. This produced better results and showed new ways researchers could expand their studies in machine learning. One of the most famous boosting algorithms is the one commonly called *AdaBoost*, or adaptive boosting [4][19]. It consists mostly of multiple base models and it is characterized by the use of weights, for both base learners and samples.

4 Methodology and Results

This chapter focuses on the methods used in this thesis to perform all experiments, and presents their results in a cohesive and understandable manner. It goes from replicating scaling laws and training models for the research to the ensembling process, the functions used, the performance variation analyzed. For the experiments done in this thesis, *nanoGPT* [10] was used, a repository specialized in small/medium-sized GPTs. It was developed by Andrej Karpathy, being a rewrite of his *minGPT*, another repository focused on facilitating the training of GPT models of smaller size, with very short and readable code.

The dependencies used in nanoGPT - and thus also in this thesis - are the following:

- PyTorch;
- Numpy;
- Transformers library from Hugging Face;
- Datasets library from Hugging Face;
- Tiktoken for OpenAI's BPE;
- WandB for logging on some experiments.

Replicating the scaling laws found in [9] proved itself to be more challenging than expected. In hindsight, the wrong configurations were used in earlier experiments, which resulted in incorrect results that did not fall under the scaling laws proposed by Kaplan et al. As a result, libraries/repositories were searched to find some way of implementing them more directly, so that our efforts could be better expended in the actual research. This way, the *Github* repository by Ali Shehper[18] was found and it proposed the exact solution we were looking for. The directory is a fork of nanoGPT that includes configuration files to replicate the experiments done by Kaplan et al [9].

We trained all of the models for 2.5×10^5 steps, with a batch size of 512 sequences of 1024 tokens. These resulted in training times of 3 to 6 days, using the *OpenWebText* dataset for training and valuation. OpenWebText is an open-source replication of the dataset used by Kaplan et al and OpenAI to train GPT-2. The models trained also were much smaller - instead of ranging from 760 to 1.5 billion non-embedding parameters [9] - ours ranged from 12 thousand to circa 9 million non-embedding parameters, showing that scaling laws are also present on a smaller scale, and facilitating training overall. The exact sizes, loss values and training times can be seen in the following table:

Non-embed. param.	Total param.	Training loss	Validation loss	Training time (h)
12288	0.83M	5.83	5.84	78.4
49152	1.69M	5.40	5.38	79.3
196608	3.48M	4.69	4.70	81.4
196608	3.48M	4.74	4.75	81.3
196608	3.48M	4.75	4.77	83.0
196608	3.48M	4.73	4.74	82.6
196608	3.48M	4.72	4.74	81.4
393216	3.67M	4.55	4.55	86.0
589824	3.87M	4.46	4.46	90.7
589824	3.87M	4.46	4.48	90.7
589824	3.87M	4.45	4.49	90.7
2359296	8.8M	3.97	3.97	141.5
9437184	22.32M	3.54	3.54	154.0

Table 4.1: All of the trained models, their non-embedding parameters, total parameters, training and validation losses. The training time is also included, in hours.

With these models, the first analysis was that of the training time. We plotted all training times and analyzed their relation with model size (non-embedding and total parameters) and also the resulting validation loss. This allowed us to see possible relations between these variables and how they relate to each other. The starting hypothesis was the simplest one: bigger models would take longer to train, and also achieve better results, which in turn would make validation loss scale with training time - longer training, smaller loss results. Together with that, the training times would be of utmost importance when it comes to ensembling. If training new bigger models would be easier than ensembling existing ones/training smaller models for ensembling, ensembles lose some of their importance as tools to reach better performance.

Afterwards, we ensembled the models, using different simple methods, to keep the experiments easy to replicate and control. The first and most important method was using the mean of the tensor results of our models. There were created ensembles of all models in combinations of up to 5 models (excluding the two biggest ones, that are used solemnly to compare performance, as ensembles reach much higher parameter counts). From all these combinations, we sampled 50 from each group of ensembles (2-model ensembles, 3-model ensembles, ..., 5-model ensembles), to facilitate our experiments, as taking predictions from all possible combinations would take way too long. These ensembles were tested against all our models and themselves and were plotted in a graph so we could directly compare their performance.

One of the tools used for this comparison was fitting the models' and ensembles' performance to power laws, such as the ones devised by Kaplan et al, comparing the observed results from our research and a possible scaling for both types of approaches. We also compared all ensembles directly with their smallest base model (which would also be their worst-performer model in our experiments) and their biggest base model (the best performer). This was done in order to examine: a) if the ensembles could improve both scenarios, showing an increase in overall performance; b) if they could improve only against the worst base models; c) the worst

case scenario if the ensembles did not improve performance at all.

In these experiments, our ensembles were a combination of random models. This posed us with the question whether ensembles of similar models would improve more or less in comparison to our random ensembles. To do that, we took 5 models with the same structure, the same amount of layers and embeddings, and the same training procedure as all other models. These models were then ensembled in all possible combinations and analyzed to see how much they could improve on average and also plotted against all other ensembles to see if they performed better overall for their size.

We also analyzed how the ensembles performed when all predictions were aggregated using *minimum* and *maximum* functions, to examine whether different aggregation methods could provide us with better results. Because the outcome was of these tests not favorable to their use, they were not tested much further. The following formulas represent the different aggregation methods used, with each "Output" representing a type of model, using the average, the minimum or maximum as aggregation *A*, and *B* representing two tensors (model outputs). In this scenario, we have two model ensembles, so only two tensors need to be aggregated - the process would be the same for more models, only the functions would have to be slightly altered.

$$\mathrm{Output}_{\mathrm{avg}} = \frac{\mathrm{A} + \mathrm{B}}{2}, \quad \mathrm{Output}_{\mathrm{min}} = \min(\mathrm{A}, \mathrm{B}), \quad \mathrm{Output}_{\mathrm{max}} = \max(\mathrm{A}, \mathrm{B})$$

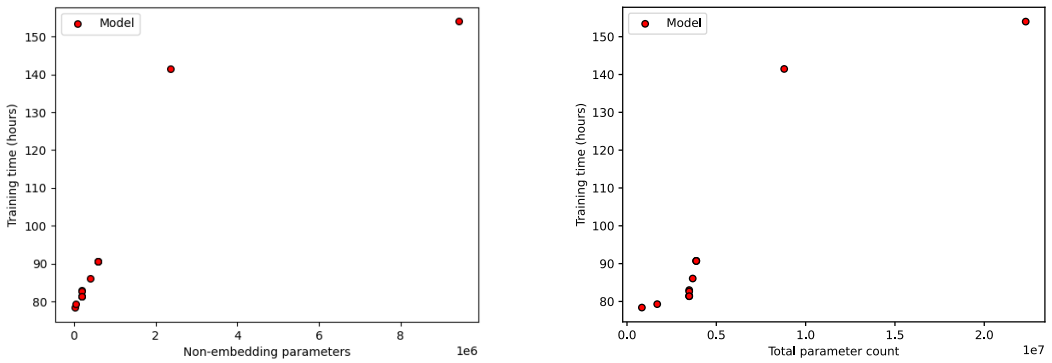
All results are compiled in the following "Results" section, mostly in plots produced using *MatPlotLib*, a visualization library for Python. They are accompanied by more detailed explanations of the experimentation process and their outcomes.

4.1 Results

4.1.1 Training time

The training of all models took between 3 days and 7 hours for the smallest model, all the way to 6 days and 10 hours for the largest one, as seen in Table 4.1. All models were trained for the exact same 2.5×10^5 steps, using the same hardware, with a batch size of 512 sequences of 1024 in the same OpenWebText dataset. One of the few differences between some of the models is their seed, which was changed for some more specific experiments that will be discussed later on.

As expected and evidenced by our data, we notice that the amount of non-embedding and or total parameters influences directly the training time of our model. Both test cases yield similar results, with what looks like an exponential relation. This increased training time could serve as an argument for the use of ensembling, as a means to improve the performance of already existing models without increasing training time, as ensembles themselves do not need any extra training.



(a) (b)

Figure 4.1: (a) Amount of non-embedding parameters in comparison to training time. (b) Total parameters in relation to training time. Model training time scales with model size when examining both non-embedding and total parameters.

Also it is worth noting, that because training time and parameter count are related, and parameter count and loss numbers also follow power laws according to Kaplan et al [9], we see that validation loss tends to decrease as the training time of a model increases, meaning that longer training times, when tied to the same training setup and configuration results in better performance from the models.

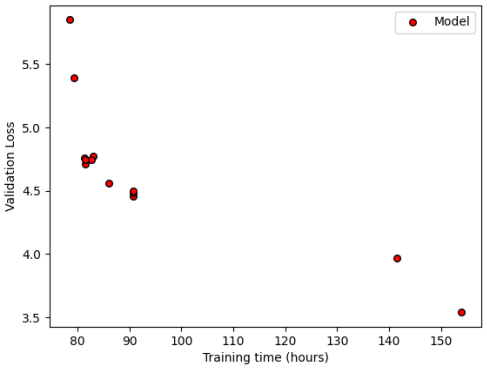


Figure 4.2: Training time in relation to validation loss. The loss decreases as the training time increases. This happens because in our scenario, where compute is fixed, longer training means bigger models and thus more performance.

Overall, our findings confirm the expected relationship between training time, parameter count, and model performance. The observed exponential trend in training duration reinforces the idea that larger models require significantly more time to train, while also benefiting from

lower validation loss. This supports previous research suggesting that parameter count and performance follow power law relationships. Additionally, our results suggest a possible use case scenario for ensembles, offering a way to enhance overall performance without resulting in additional training costs, when ensembling already existing models - but we explore this in full, in the following subsections.

4.1.2 Replicating parameter scaling laws

For our future ensembling experiments, it was important to be able to replicate scaling laws, as we want to analyze if we can find better performance with ensembling than with regular models, and we use the scaling laws as a guide/measure, so we can extrapolate our results to a bigger scale. To replicate the scaling laws found in [9], we trained a total of 13 models of varying size, using the same configuration as to limit the variables as much as possible. After the training, we got validation loss values from 5.85 to 3.54, with the biggest loss belonging to the smallest of our models and vice-versa.

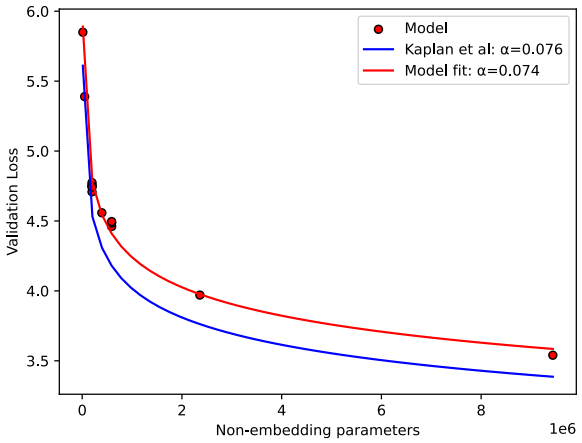


Figure 4.3: Results from our model training. Each dot represents a single model, its loss, and the number of non-embedding parameters. The lines represent our fit (in red) to power laws in comparison to Kaplan et al's fit (in blue). Both have similar values, proving that scaling laws are present even on smaller scale.

Once all models were plotted (figure 4.3), the values formed a clear power law relation between loss and model size, more specifically non-embedding parameters, as expected. Considering the following formula: $L(N) = \left(\frac{N_c}{N}\right)^{\alpha}$ [9], we got very similar results to Kaplan et al, reaching an $\alpha = 0.074$, whilst their experiments resulted in an $\alpha = 0.076$. We attribute the small variation to the different dataset that was used - as ours was an open-source version from theirs - and possibly random chance. To calculate these numbers, we used the *polyfit* function from Numpy, which automatically does the polynomial fitting for our set of points (models). To do so, we used this function to find a first-degree polynomial that fit our data when in logarithmic space.

In this thesis, we focused on replicating only the non-embedding parameter scaling laws, as they were of prime importance when analyzing our ensembling results. Replicating both compute power laws and dataset size was for our scenarios irrelevant, as we did not let the dataset and compute power limit our experiments. In the end, we got very close results from Kaplan et al, thus proving that scaling laws are also present even on a much smaller scale

than the one their research examined originally: their models ranged from 760 million non-embedding parameters all the way to 1.2 billion. Ours ranged from 12 thousand to 9 million parameters - even our biggest models are dwarfed by their smallest.

4.1.3 Ensembling

After replicating the parameter scaling laws and having a substantial amount of models to test with, we started ensembling the models. Different experiments were executed, which shall be explained in this subsection. For our first test, we took all possible combinations of our models. We divided them into groups, being all possible 2-model combinations, 3-models, and up to 5-model combinations. From each of these, we sampled 50 ensembles randomly, so that testing became a lot faster, as trying all possible combinations took way too long. To combine the results of each model, the mean was the chosen method - so for all ensembles, the mean of the predictions of all their base models was taken and analyzed as the final prediction.

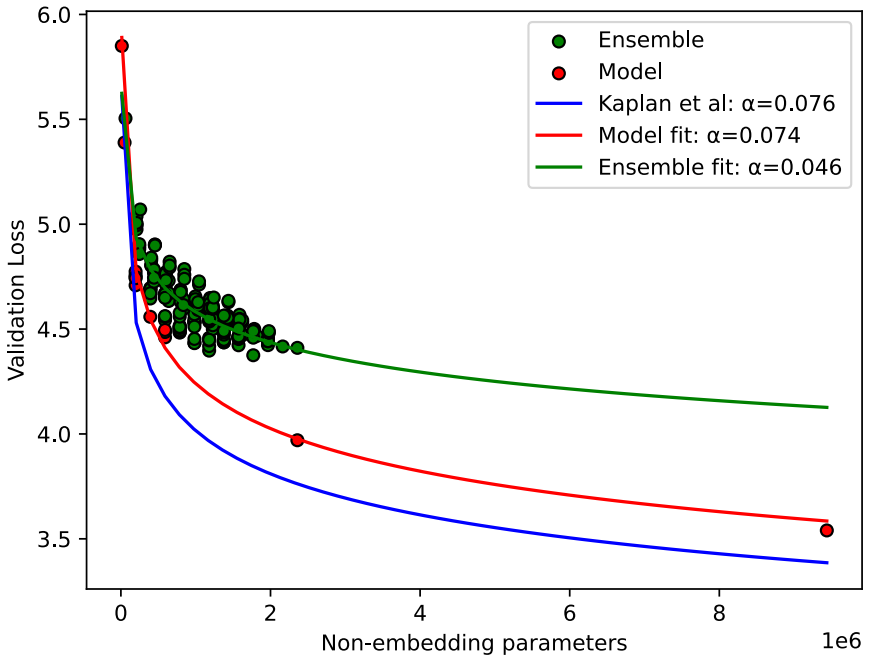


Figure 4.4: Models and ensembles plotted together. Lines were drawn to indicate the fit (possible progression) of models (red), ensembles (green), and the power law provided by Kaplan et al (blue).

The results were quite interesting, with ensembles that could present better results than all of their base models, but the scaling of ensembles turned out to be much less beneficial than the scaling of actual models. Using the same logic to calculate a power law using our ensemble

results as a base, we see that this power law differs strongly from our model one. Having an $\alpha = 0.046$, and thus a much worse scaling over time. Assuming this line would stay true for bigger and bigger ensembles, we see that training models to the same size would always outperform our ensembles. But in turn, it also poses the possibility of better results without any more training, even if with diminishing returns - considering the models to ensemble are already available.

When comparing the ensemble's performance with their smallest base model, we see that almost in every sampled case we see direct improvement - from that we see that ensembling smaller/worse models with bigger and better-performing ones results in better results almost every time, when averaging their outputs.

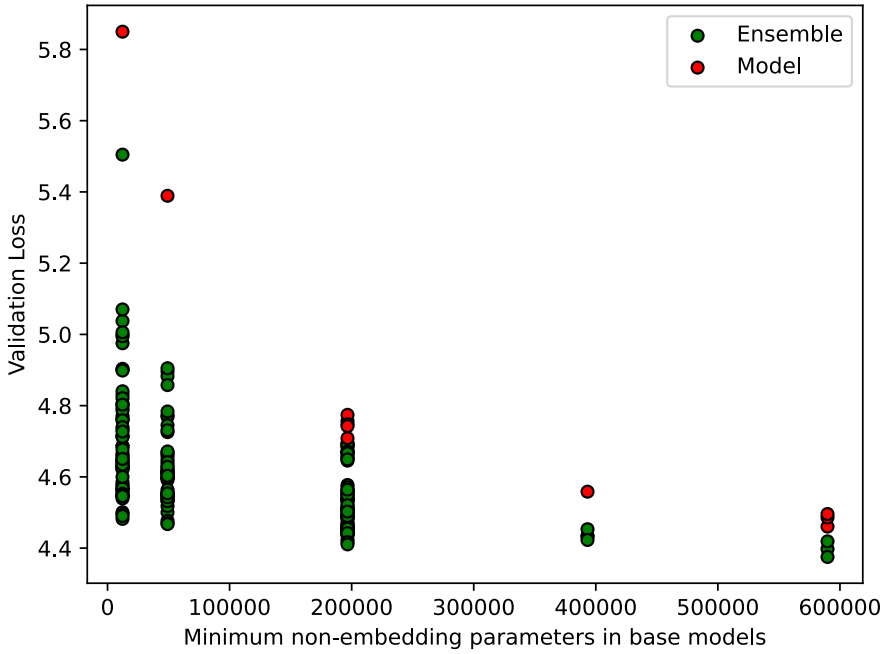


Figure 4.5: Comparing ensembles with their smallest base model. In every case, the ensembles outperform the models, following scaling laws: more parameters, more performance.

In contrary, when analysing the validation loss results from our models taking only the biggest model in the ensemble, we see that the ensemble's performance is closely tied to them, rarely outperforming the base models. This still proves worthy of consideration, as in some cases it still provides better performance, even if marginally.

In summary, we see that ensembling different models always results in a better performance when comparing them with their worst-performer base model. It can also result in better outputs than their best performer base model, but only in a few cases. This shows that there

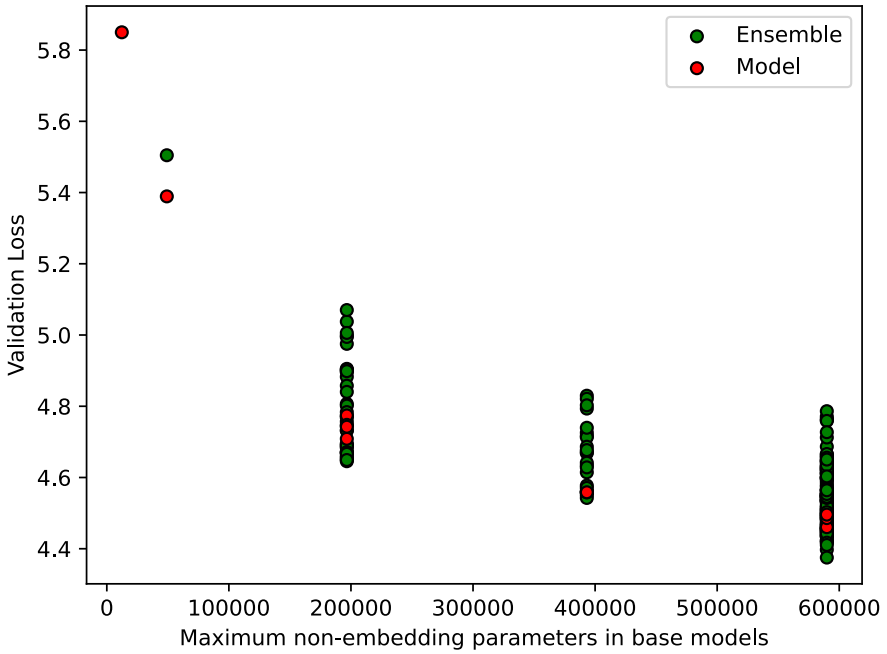


Figure 4.6: Comparing ensembles with their biggest base model. In some cases, ensembles outperform the models, but not in all situations, showing that the performance of the biggest base learner still is a very important variable.

is a benefit in ensembling, as we can increase performance on average, without any extra training - though marginally. Even then, it is worth noting, that training bigger models results in a better performance to parameters ratio, as our ensembles are fit to a much flatter line of improvement, based on our observations.

4.1.4 Ensembling similar models

While in the previous experiment, we tested all possibilities and combinations, for this we tried ensembling only models with the same number of layers and embeddings - the only difference in training being their seed so that the end results would not be completely equal. This resulted, as expected, in very similar models, with very similar outputs. The main hypothesis would be that similar models would provide better outcomes when ensembled together than totally different models. In previous experiments we were able to see that usually, the bigger models would be the driving factor on the loss of the ensembles, so we also wanted to examine whether ensembling very similar models would yield a better overall result.

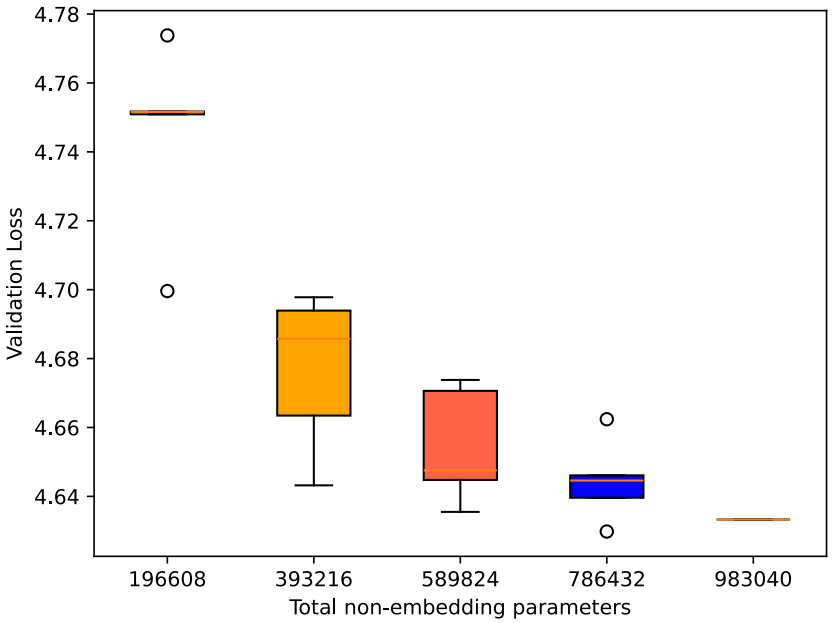


Figure 4.7: Ensembling results. As all models have the same amount of parameters, the first column represents the results of the models themselves. The second is the result of all 2-model combinations, and so forth, up to the only possible 5-model ensemble.

The models used all had 4 layers with 64 as the number of embeddings. Because of that, all of them had 196608 parameters total, with their ensembles always having a multiple of this number. The variation of loss between the models themselves was already apparent after training. The best model scored 4.69 in validation loss, whilst the worst scored around 4.77. A not so great difference when analysing different models, but considering all of these were trained with the same conditions and parameters, this takes on more importance and we can clearly note that there is some variation when it comes to the training of nearly identical large language models.

This variation was also directly transferred to our ensembles. Looking at the data, it seems

apparent that ensembles that utilize the best performing model outperform the others, making the ensembles themselves have a strong variation within their ranks. Nonetheless, it is very important to note, that in our experiments, even the worst ensemble practically equaled the best model, and when analyzing our medians, we see that increasing the number of models in an ensemble improves performance. Even then, when analyzing our data, we did not find any substantial increase in performance when using models with similar/equal structures. When plotted with all other ensembles, we note that they are not the best performers for their size, as models and other ensembles beat them with some margin.

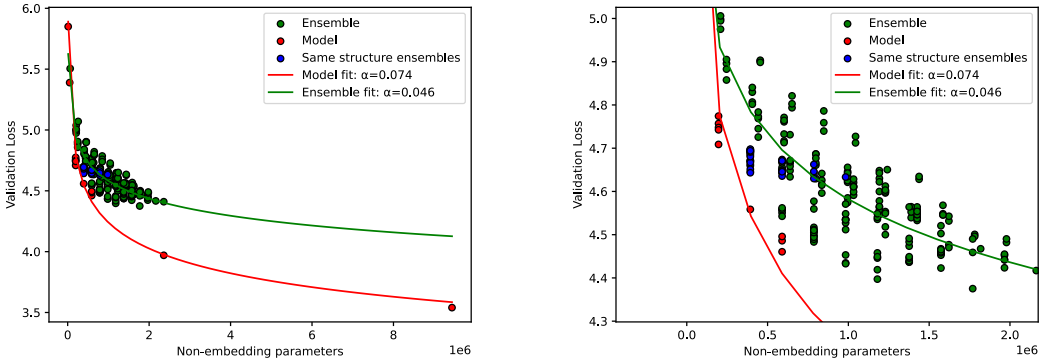
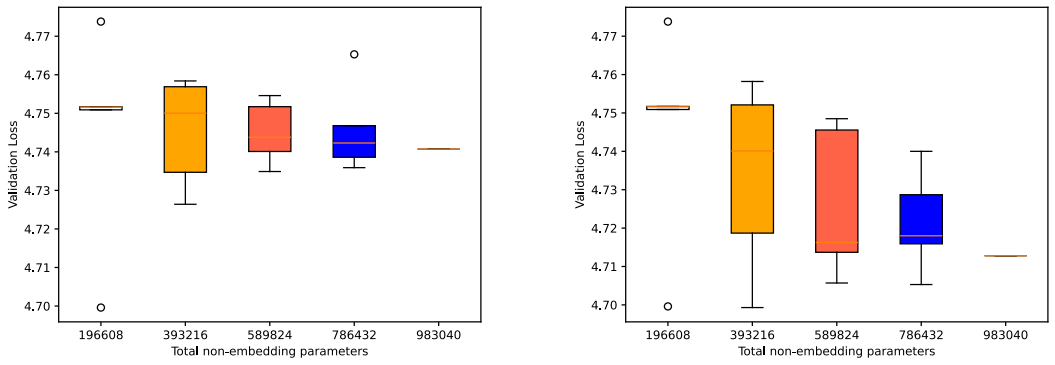


Figure 4.8: (a) All models and ensembles plotted together, the x-axis being their non-embedding parameters, and y-axis their validation loss. The ensembles analyzed in this chapter (same structure) are highlighted in blue. (b) Same plot as on the left, but zoomed in for better understanding.

When analyzing other methods of aggregating results, more specifically *minimum* and *maximum* from all base model predictions, we noted a different outcome. In both scenarios, there were little to no performance benefits, and the results looked a lot more random throughout all predictions. The *minimum* function as aggregator showed little improvement when increasing ensemble size, with the best median result from the ensembles barely beating the mean from the models themselves.

When using the *maximum*, even the best-performing ensemble only results in a validation loss on par with the best-performing model, barely beating it by 0.003 points, which because of randomness in our predictions, is completely negligible. The medians still showed some improvement with bigger ensembles, but the results were overshadowed by the performance when using the mean as the aggregator.



(a) (b)

Figure 4.9: (a) Using the minimum as aggregator for all model outputs delivers worse results than our best model, and almost negligible improvement on average. (b) Using the maximum of all model outputs gives us better results than the minimum on average, but all ensembles still perform worse than our best-performing model.

5 Conclusion

In this thesis, we analysed the performance effects of ensembling large language models. To do that, we started by replicating and analyzing the scaling laws defined by Kaplan et al [9], in a search for a better way to scale models, using ensembles, maybe even breaking these laws. We trained 13 models, with different configurations when it came to size, number of layers, and other factors, while keeping dataset and compute power as constant as possible throughout training. This gave us the possibility, not only of comparing our results directly with the aforementioned law but also of further research by ensembling the models and examining the outcome.

We observed that our models closely followed the laws established by Kaplan et al, whilst the ensembling results did not. They did not scale as well with their size, resulting in a less optimal performance-to-size ratio. To mitigate that, we looked into other possible variables, such as ensembling only models with similar structures, and using different aggregating methods. These, however, also did not improve our models' performance by much, maintaining our outcome. Model structure did not seem to matter to performance: similar models did not ensemble better than randomly picked ones. The different aggregation methods also did not seem to provide us with better results than ensembling by using the average of all predictions. We noticed that in all our results, when using random ensembles and the average for ensembling, the ensembles always outperformed their worst base model (when using validation loss as our metric), but only in some cases outperformed their best base models.

Thus considering a scenario similar to ours in this thesis, where the models are already available, ensembling might be a useful tool in pushing the performance to slightly better levels without training new models. Even then, it is worth noting that models trained to the same size always perform better than the ensembles based on our observations, so training models directly with ensembling in mind seems like a very suboptimal use of training resources. There is still space for more research, as the possibilities when it comes to ensembles are very abundant, and due to time constraints, we could not explore most of them. One of the possible areas worth exploring is training the ensembles themselves for some epochs/steps, or implementing more complex ensembling methods, such as AdaBoost [4]. These might be the breakthroughs needed to overcome the scaling laws proposed by Kaplan et al and reach new heights in the performance of modern large language models.

Bibliography

[1] [n. d.]. Robust and Efficient Millimeter-Wave Massive MIMO Hybrid Precoding Architecture based on the Perceiver Neural Network - Scientific Figure on ResearchGate. https://www.researchgate.net/figure/Basic-Transformer-model-architecture_fig2_365039864 Accessed: 2025-02-13].

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

[3] Josh Achiam et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[4] Yoav Freund, Robert E Schapire, et al. 1996. Experiments with a new boosting algorithm. In *icml*, Vol. 96. Citeseer, 148–156.

[5] Philip Gage. 1994. A new algorithm for data compression. *The C Users Journal* 12, 2 (1994), 23–38.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2017. The elements of statistical learning: data mining, inference, and prediction.

[7] Tin Kam Ho. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20, 8 (1998), 832–844. https://doi.org/10.1109/34.709601

[8] Jiawei Jiang, Bin Cui, and Ce Zhang. 2022. *Distributed Machine Learning and Gradient Optimization.* Springer. 5–12 pages.

[9] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG] https://arxiv.org/abs/2001.08361

[10] Andrej Karpathy. 2022. nanoGPT. https://github.com/karpathy/nanoGPT Accessed: 2025-03-03.

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Simon Klüttermann, Jérôme Rutinowski, and Emmanuel Müller. 2024. The Phenomenon of Correlated Representations in Contrastive Learning. In *2024 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN60899.2024.10649913

[13] Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. 2021. Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP. arXiv:2112.10508 [cs.CL] https://arxiv.org/abs/2112.10508

[14] OpenAI. 2024. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/ Accessed: 2025-03-03.

[15] Alec Radford and Karthik Narasimhan. 2018. Improving Language Understanding by Generative Pre-Training. https://api.semanticscholar.org/CorpusID:49313245

[16] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[17] Robert E Schapire. 1990. The strength of weak learnability. *Machine learning* 5 (1990), 197–227.

[18] Ali Shehper. 2022. Scaling Laws. https://github.com/shehper/scaling_laws Accessed: 2025-03-03.

[19] Leonardo Vanneschi and Sara Silva. 2023. *Ensemble Methods*. Springer International Publishing, Cham, 283–288. https://doi.org/10.1007/978-3-031-17922-8_11

[20] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[21] Haifeng Wang, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. 2023. Pre-Trained Language Models and Their Applications. *Engineering* 25 (2023), 51–65. https://doi.org/10.1016/j.eng.2022.04.024

[22] Zichong Wang, Zhibo Chu, Thang Viet Doan, Shiwen Ni, Min Yang, and Wenbin Zhang. 2024. History, Development, and Principles of Large Language Models-An Introductory Survey. arXiv:2402.06853 [cs.CL] https://arxiv.org/abs/2402.06853

[23] Jonathan J Webster and Chunyu Kit. 1992. Tokenization as the initial phase in NLP. In *COLING 1992 volume 4: The 14th international conference on computational linguistics*.

[24] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022. Emergent

Abilities of Large Language Models. arXiv:2206.07682 [cs.CL] https://arxiv.org/abs/2206.07682

Eidesstattliche Versicherung

(Affidavit)

Allebrandt Werlang, Arthur
Name, Vorname
(surname, first name)

236512
Matrikelnummer
(student ID number)

☒ Bachelorarbeit
(Bachelor's thesis)

☐ Masterarbeit
(Master's thesis)

Titel
(Title)

Neural scaling laws: the impact of large language model ensembles on their performance

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Paverama-RS, BR, 31. März 2025
Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Paverama-RS, Brasilien, 31. März 2025
Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**