Trends in LLMs & scaling laws

Stanislav Fedotov, Practical Generative AI course

January 2024

1 Trends in LLMs, as of late January 2024

I would say, that there are two important trends now: efficiency and multimodality. You can learn more about multimodal LLMs in the separate long read, so in this one we'll mostly focus on efficiency.

There are several axes along which we could measure efficiency, including:

- Sheer model size (see Small models),
- Efficiency at inference stage (see Mixture of Experts; we'll also discuss approaches like inference-time quantization in Module 2),
- Efficiency at longer context (see Non-transformer models, although they are also efficient in other ways),
- Getting more value from available data (see Squeezing more from meager data)

2 Small models

While ChatGPT is cool, only very rich companies like OpenAI or Google can afford training something of this scale and many wouldn't be able to afford even deploying it for inference. And, honestly speaking, for many tasks a fine tuned small LLM could work better than ChatGPT.

So, while OpenAI and Meta promise us the miracles of GPT-5 and LLaMA 3, the hopes of common ML engineers often lie with smaller models, such as Phi-2, Mixtral and others. Each model has its own secret of success. Let's look at some of them.

2.1 Training on lots of data

You already know about Mistral 7B that outperformed LLaMA 2-13B across all evaluated benchmarks (and even the best released 34B model) and approached the coding performance of Code-Llama 7B without sacrificing performance on non-code related benchmarks.

Creators of small models trained on vast data partially drew inspiration from the 2023 research on the **scaling laws**.

Before the Chinchilla paper it was widely believed that the size of LLM should grow much faster than the size of the training data, which explains why people tried to make LLMs larger and larger. After Chinchilla, the table turned.

See more about Chinchilla paper in the Scaling Laws section.

2.2 Careful training data curation

You already know how creation of RefinedWeb (CommonCrawl with almost 90% of data cleaned out) powered **Falcon LLM**.

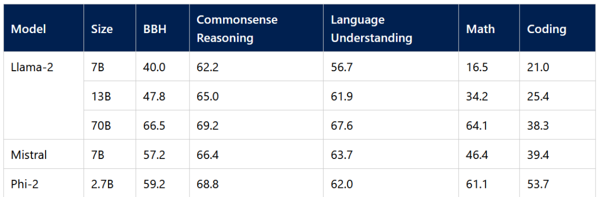
The same idea inspired Microsoft Research in creation of the Phi LLM series. **Phi-1**, an LLM for code generation, launched around mid-2023, had just 1.3B parameters, but achieved quality rivaled only by GPT-4 and WizardCoder at that time. In the paper Textbooks are all you need the authors argued that cleaning out low-value training data and introducing high-quality synthetic data would be beneficial. What they eventually included in the training dataset was:

- A filtered code-language dataset (consisting of about 6B tokens), which is a subset of The Stack and StackOverflow. To create the filtering, the authors first annotated the quality of about 100k samples GPT-4 prompted to “determine its educational value for a student whose goal is to learn basic coding concepts”. Then, they used this annotated dataset to train a random forest classifier (wow!) on output embedding from a pretrained codegen model as features.

- A synthetic textbook dataset consisting of < 1B tokens of GPT-3.5 generated Python textbooks.

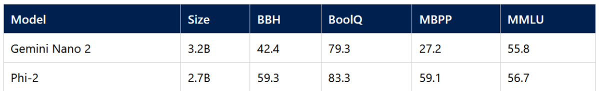
- A small synthetic exercises dataset consisting of ~ 180M tokens of Python exercises and solutions.

In December 2023, Miscrosoft published an even more capable LLM, Phi-2. It has 2.7B parameters, but outperforms powerful LLMs such as Mistral-7B and LLaMA-2-13B:



Model	Size	BBH	Commonsense Reasoning	Language Understanding	Math	Coding
Llama-2	7B	40.0	62.2	56.7	16.5	21.0
	13B	47.8	65.0	61.9	34.2	25.4
	70B	66.5	69.2	67.6	64.1	38.3
Mistral	7B	57.2	66.4	63.7	46.4	39.4
Phi-2	2.7B	59.2	68.8	62.0	61.1	53.7

Table 1. Averaged performance on grouped benchmarks compared to popular open-source SLMs.



Model	Size	BBH	BoolQ	MBPP	MMLU
Gemini Nano 2	3.2B	42.4	79.3	27.2	55.8
Phi-2	2.7B	59.3	83.3	59.1	56.7

Table 2. Comparison between Phi-2 and Gemini Nano 2 Model on Gemini’s reported benchmarks.

The training data of Phi-2 was a mixture of:

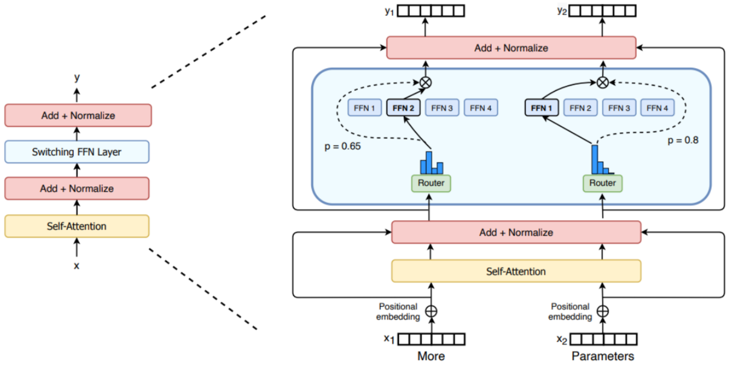
- Synthetic data specifically created to teach the model common sense reasoning and general knowledge, including science, daily activities, and theory of mind, among others.
- Carefully selected web data that is filtered based on educational value and content quality.

Since January 6, 2024, Microsoft released Phi-2 under the MIT Open-Source License.

2.3 Mixture of experts

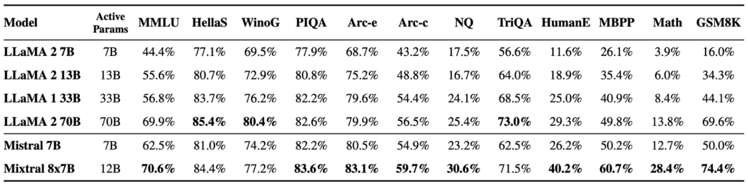
Mixtral-8x7B seems to be huge, it has 46.7B total parameters. However, it only uses 12.9B parameters per token at inference time, like the smallest LLaMA-2.

This is achieved thanks to Mixture of Experts architecture:



In short, we replace feedforward layer of the Transformer block by several feedforward ”experts” with a special ”router” function deciding to which of the experts to send the next data point. We'll look at the formulas more closely in Module 2, but if you're already curious, you can check this blog post.

This ”hidden” complexity allows Mixtral to beat many other models while remaining as efficient as a 13B LLM:



Model	Active Params	MMLU	HellaS	WinoG	PIQA	Arc-e	Arc-c	NQ	TriQA	HumanE	MBPP	Math	GSM8K
LLaMA 2 7B	7B	44.4%	77.1%	69.5%	77.9%	68.7%	43.2%	17.5%	56.6%	11.6%	26.1%	3.9%	16.0%
LLaMA 2 13B	13B	55.6%	80.7%	72.9%	80.8%	75.2%	48.8%	16.7%	64.0%	18.9%	35.4%	6.0%	34.3%
LLaMA 1 33B	33B	56.8%	83.7%	76.2%	82.2%	79.6%	54.4%	24.1%	68.5%	25.0%	40.9%	8.4%	44.1%
LLaMA 2 70B	70B	69.9%	**85.4%**	**80.4%**	82.6%	79.9%	56.5%	25.4%	**73.0%**	29.3%	49.8%	13.8%	69.6%
Mistral 7B	7B	62.5%	81.0%	74.2%	82.2%	80.5%	54.9%	23.2%	62.5%	26.2%	50.2%	12.7%	50.0%
Mixtral 8x7B	12B	**70.6%**	84.4%	77.2%	**83.6%**	**83.1%**	**59.7%**	**30.6%**	71.5%	**40.2%**	**60.7%**	**28.4%**	**74.4%**

Mixtral-8x7B-Instruct-v0.1 is now a top model at Chatbot Arena among all non-proprietary models.

3 Squeezing more from meager data

It you're not OpenAI or Google, you're constantly lacking something, and it's not only compute. You don't have much data either.

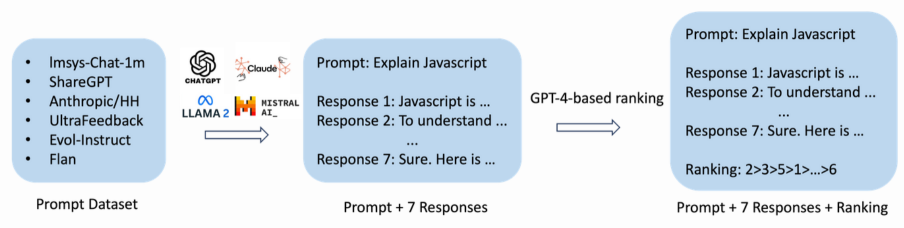
3.1 Getting data from the existing powerful models

This was done by the authors of Alpaca: they prompted GPT to get more instruction data. It is also widely used in Multimodal LLMs (see the separate long read). And there is much more.

Another area where existing powerful LLMs can help is providing feedback for the ~~RLHF~~ RLAIF fine tuning stage. For example, it is done in Starling-7B. This model was fine tuned from Openchat 3.5 using the Nectar dataset containing 183K chat prompts.

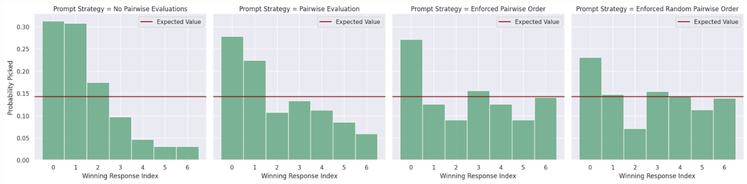
Nectar was gathered as follows:

- For each prompt, the authors got 7 responses distilled from various models like GPT-4, GPT-3.5-instruct, GPT-3.5-turbo, Mistral-7B-Instruct, Llama2-7B.


lmsys-Chat-1m
ShareGPT
Anthropic/HH
UltraFeedback
Evol-Instruct
Flan
Prompt Dataset
CHATGPT
Claude
LLAMA 2
MISTRAL AI_
Prompt: Explain Javascript
Response 1: Javascript is ...
Response 2: To understand ...
...
Response 7: Sure. Here is ...
Prompt + 7 Responses
GPT-4-based ranking
Prompt: Explain Javascript
Response 1: Javascript is ...
Response 2: To understand ...
...
Response 7: Sure. Here is ...
Ranking: 2>3>5>1>...>6
Prompt + 7 Responses + Ranking

*Illustrating the creation process of Nectar, a 7-wise comparison dataset for RLAIF.

- GPT-4 was used to rank the responses. A challenging aspect was overcoming Positional Bias: GPT-4 clearly favored responses in the first and second positions when it was simply asked to rank responses without additional reasoning.



*The positional bias of GPT-4-based ranking.

To address this, the authors instructed GPT-4 to first conduct pairwise comparisons for all response pairs before compiling a 7-wise ranking.

The authors train RLHF with this 7-wise preference data, instead of ordinary pairwise ones. For this, they use Plackett-Luce Model. It's more complicated than the Terry-Bradley model used in original RLHF, and I'll try to include it in the math of RLHF&DPO in the second module. If you want to understand it better right now, you can find an digestible explanation here.

They also tried DPO, but it didn't work well, probably because Openchat 3.5 was itself fine tuned using an RL algorithm.

Starling is really good. Its utmost drawback is, thought, that it is only available for non-commercial use :(

3.2 Training for more epochs

Initially, most LLMs were pre-trained using only one pass over a dataset, and Chinchilla scaling laws were discovered exactly for this scenario. However, it turns out that good old multi-epoch training can still be useful: for the first 4 epochs it can be as good as training on 4x of unique data, see Scaling laws with constrained datasets section for details.

3.3 Using RL to mitigate the lack of data

The idea of RL is to train a model via feedback for its outputs. In a sense, it doesn't require training data, just a reward model of sorts and lots of patience. There were several papers recently when RL is used for tuning LLMs. I'll describe the ideas quickly, and the math of it will wait till Module 2.

Self-Play Fine-Tuning

Self-Play Fine-Tuning Converts Weak Language Models to Strong Language Models addresses the question of whether a small LLM is able to fully benefit from SFT data. The authors take zephyr-7b-sft-full, a fine-tuned LLM based on Mistral-7B, and show that:

- If we try to fine tune it further on its own SFT dataset, we don't improve it much and can even diminish the evaluation scores.
- Still, LLM's completions underperform in comparison to the ground truth completions from the dataset, so it seems that we could probably squeeze some more information from this data.

The authors suggest a new approach, **Self-Play Fine-Tuning (SPIN)**, which involves interaction of two "players":

1. The main player is a score function that tries its best to distinguish between the real SFT data distribution and the distribution of LLM's outputs (this score function resembles the one of DPO).
2. The opponent is the LLM itself which tries its best to mimic the distribution of real data.

Experiments show that SPIN can significantly improve the LLM's performance across a variety of benchmarks and even outperform models trained through DPO on a dataset with accepted/rejected labels given by GPT-4.

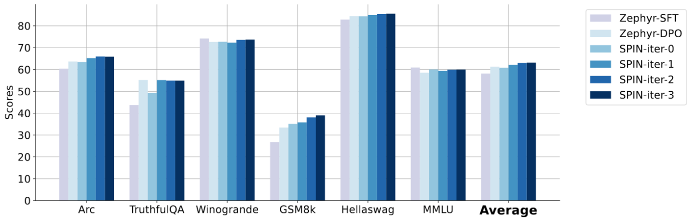


Figure 3: Performance comparison with DPO training across the six benchmark datasets. Self-play at iteration 0 achieves comparable performance to DPO training with 62k new data. At iteration 1, self-play has already surpassed DPO training on the majority of datasets.

Note however, that SPIN is not a replacement for RLHF/DPO, because it's goal is completely different: while SPIN makes an LLM produce more "likely" completions, it doesn't introduce any kind of human preferences, so RLHF/DPO step can still be needed after it.

Self-Rewarding Language Models

Link to the paper

We already know that each LLM is secretly its own reward model (it's the DPO, Direct Preference Optimization paper). The authors of self-rewarding models go even further and suggest generating data for DPO with the same model and even using the same LLM to rank potential completion for judging which is accepted and which is rejected.

Namely, they propose the following algorithm:

1. Sample a new prompt from instruction fine tuning dataset,
2. Generate candidate responses for it,
3. Use the LLM-as-a-Judge ability of our same model to evaluate the candidate responses with scores from 0 to 5,
4. Choose accepted and rejected pair for DPO by taking the highest and lowest scoring responses. If they have the same score, disregard this pair.

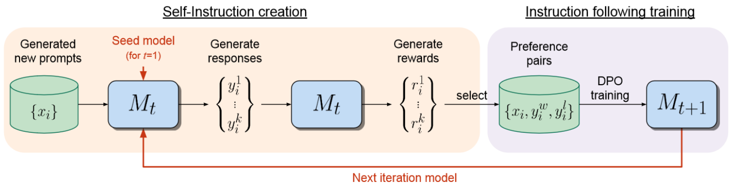


Figure 1: **Self-Rewarding Language Models.** Our self-alignment method consists of two steps: (i) *Self-Instruction creation*: newly created prompts are used to generate candidate responses from model M_t, which also predicts its own rewards via LLM-as-a-Judge prompting. (ii) Instruction following training: preference pairs are selected from the generated data, which are used for training via DPO, resulting in model M_{t+1}. This whole procedure can then be iterated resulting in both improved instruction following and reward modeling ability.

For example if we have really scarce preference data, we can start by fine tuning on it, then prompt the LLM to create new training data for the next stage etc.

The results seem quite nice:

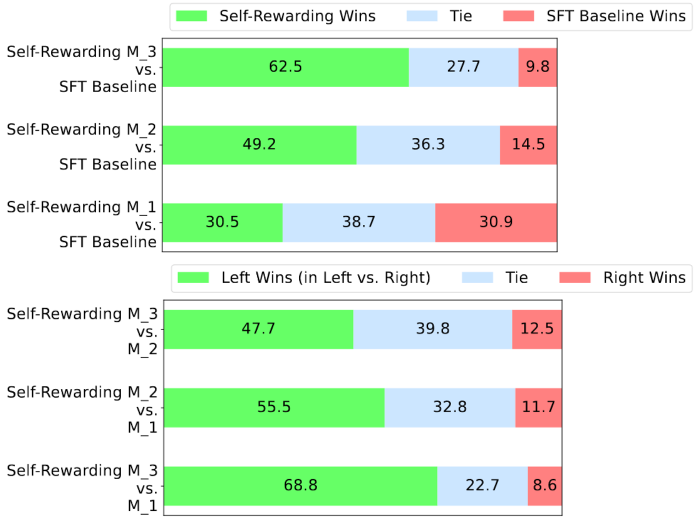


Figure 3: **Instruction following ability improves with Self-Training:** We evaluate our models using head-to-head win rates on diverse prompts using GPT-4. The SFT Baseline is on par with Self-Rewarding Iteration 1 (M_1). However, Iteration 2 (M_2) outperforms both Iteration 1 (M_1) and the SFT Baseline. Iteration 3 (M_3) gives further gains over Iteration 2 (M_2), outperforming M_1, M_2 and the SFT Baseline by a large margin.

4 Non-tranformer models are gaining ground

Transformers have reigned supreme in NLP for quite a long time, but now they seem to have a worthy competitor — **State Space Models**.

We discuss this type of models in Week 8, but here are very short takeaways:

- State space model-based LMs combine convolutional and recurrent principles, which allow them to train efficiently (as CNNs) and be quick at inference (as RNNs),

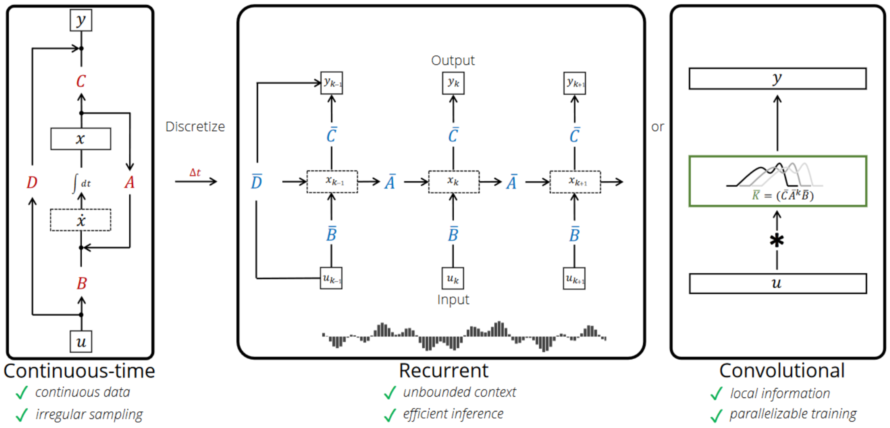
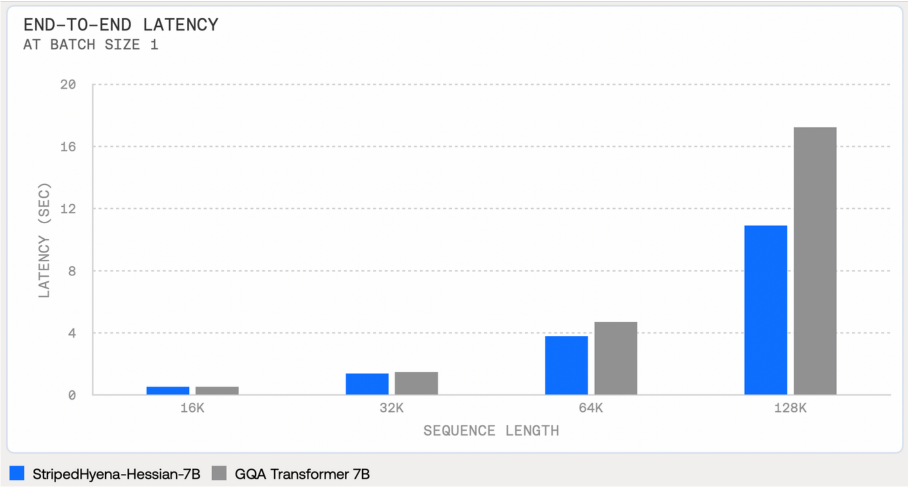


Figure 1: (**Three views of the LSSL**) A **Linear State Space Layer** layer is a map $u_t \in \mathbb{R} \to y_t \in \mathbb{R}$, where each feature $u_t \mapsto y_t$ is defined by discretizing a state-space model A, B, C, D with a parameter Δt. The underlying state space model defines a discrete recurrence through combining the state matrix A and timescale Δt into a transition matrix $\overline{A}$. (**Left**) As an implicit continuous model, irregularly-spaced data can be handled by discretizing the same matrix A using a different timescale Δt. (**Center**) As a recurrent model, inference can be performed efficiently by computing the layer *timewise* (i.e., one vertical slice at a time $(u_t, x_t, y_t), (u_{t+1}, x_{t+1}, y_{t+1}), \ldots$), by unrolling the linear recurrence. (**Right**) As a convolutional model, training can be performed efficiently by computing the layer *depthwise* in parallel (i.e., one horizontal slice at a time $(u_t)_{t\in[L]}, (y_t)_{t\in[L]}, \ldots$), by convolving with a particular filter.

- When coded well, they are more efficient, than transformers, especially for long contexts.



- As transformers, they can also be used for pictures, where they exploit the same idea of breaking an image into patches.

There are several open source models of this type that you can already use in your projects, inluding:

- StripedHyena-7B(chat version at Hugging Face),
- and an increasingly popular Mamba (at Hugging face: link).

5 Scaling laws for LLMs

If you want to train an LMM from scratch, there are four quantities that you need to worry about:

1. **Number of parameters**. The larger it is, the more potent can the model potentially be, but the less feasible it can be in terms of latency and memory requirements.
2. **Volume of data** you have for training. Typically, the more the better, but smaller models can fail to leverage too large text corpora. Data volume is usually measured in tokens.
3. **Compute availability**. It's measured in FLOPs (floating-point operations) and roughly tells how many GPU hours you're ready to spend for training until you run out of money or get happily retired.
4. **Loss** (cross entropy), which is a proxy

You can ask yourself questions like:

- How large a model is it reasonable to train given certain amounts of tokens/compute?
- Given a fixed amount of available FLOPs, what is the optimal [model size]/[data volume] trade-off? Obviously, these two things compete for compute. So, should we train larger models on fewer data or smaller models on more data?

Throughout this section we will use the following notation:

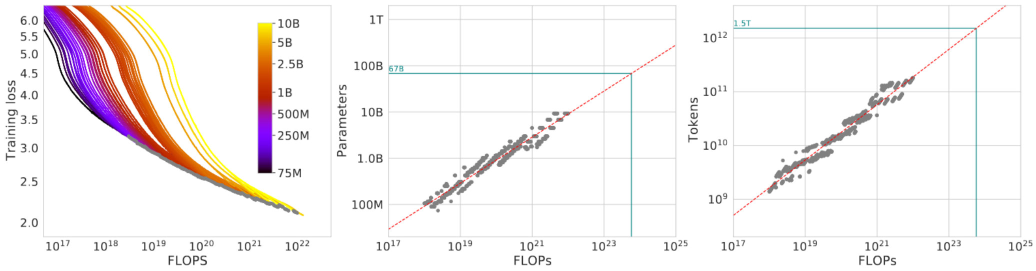
- N is the number of model parameters,
- D is the training dataset size in tokens,
- L is the cross entropy loss which estimates the quality obtained with training.

5.1 Chinchilla paper

Training Compute-Optimal Large Language Models by DeepMind.

The authors trained transformers with various N and D with specific batch size and learning rate schedule and recorded loss values. An important thing here is that all tokens are unique, so we make not more than one pass over the training datasets. The authors analyzed the gathered data under the following three approaches:

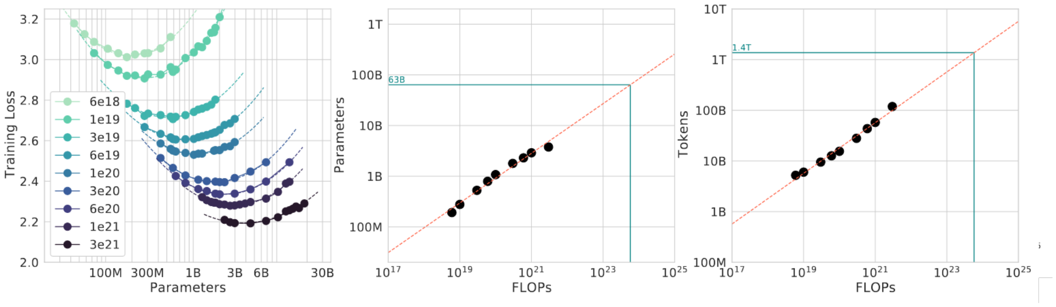
Approach 1: Fix model sizes and vary number of training tokens. What they get:



On the left you see learning curves for models with different number of parameters. The grey points are those with optimal losses for a fixed FLOPs budget.

On the right, you see the same grey dots, but now instead of loss we plot number of parameters or number of tokens. Note that each point of a learning curve corresponds to a moment of the training process and to a number of tokens processed so far. The dashed line estimates the optimal numbers of parameters and tokens given a fixed FLOPs budget.

Approach 2: isoFLOP profiles.



On the left each U-shaped curve corresponds to a fixed FLOPs budget. The higher the budget, the lower is the loss it allows to achieve. Each curve has a minimum, the optimal number of parameters with this budget.

On the right, we plot the numbers of parameters and tokens for the minimums of U-shaped curves. The dashed line is fitted through these points.

Approach 3: math model.

The authors model the connection between N, D and the minimal loss L that can be achieved by an LLM of size N trained on a dataset of size D using the following formula:

$$L(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}.$$

Justification (caution: math inside). Let's introduce the following notation:

- f^* is the hypothetical best LLM which minimizes L over all imaginable texts in the universe. Mathematically,

  $$f^* = \text{argmin}_f \mathbb{E}\, L(y, f(x)),$$

  where expectation is taken over all imaginable pairs (true completion, predicted completion).
- f_N is the transformer-based LLM of size N which minimizes L over all imaginable texts in the universe.
- $\overline{f}_{N,D}$ is the best transformer of size N which is trained on the most favorable set of D tokens with gradient descent with specific batch size.

Obviously, f^* and f_N^* only exist in our dreams, while $\overline{f}_{N,D}$ is almost real. We can decompose $L(N, D)$ as:

$$L(N, D) = L(f_{N,D}) = L(f^*) + (L(f_N) - L(f^*)) + \left(L(\overline{f}_{N,D}) - L(f_N)\right)$$

The first summand $L(f^*)$ is the minimal possible loss value, the "entropy of a natural text". This will be the E term.

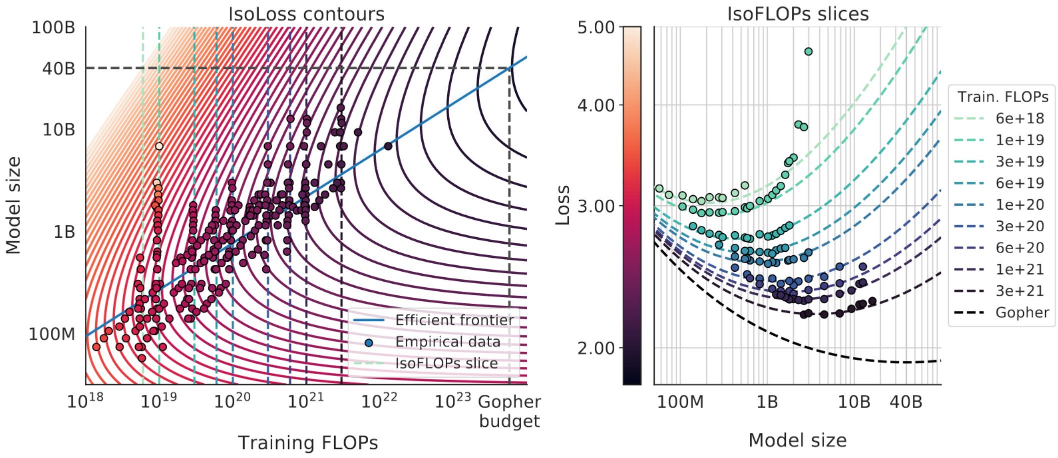
The second summand $(L(f_N) - L(f^*))$ tells us how much an ultimate transformer of size N is worse than f^* in terms of loss, something like an unavoidable error value of transformers. It only depends on N. Moreover, on the set of two-layer neural networks, it is expected to be proportional to $\frac{1}{N^{1/2}}$ (see this paper), and this motivates the authors of Chinchilla to estimate the second term as $\frac{A}{N^\alpha}$.

The third summand $\left(L(\overline{f}_{N,D}) - L(f_N)\right)$ measures how much a transformer of size N trained in a particular way on a dataset of size D is worse than a hypothetical optimal transformer of size N. We can assume that it only depends of D, and it can be shown that it is lower-bounded by $\frac{1}{D^{1/2}}$, so the authors estimate it as $\frac{B}{D^\beta}$.

Fitting the coefficients. Using the data they gathered, the authors fit the following model:

$$L(N, D) = 1.69 + \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.28}}$$

Using this, they plot the following:

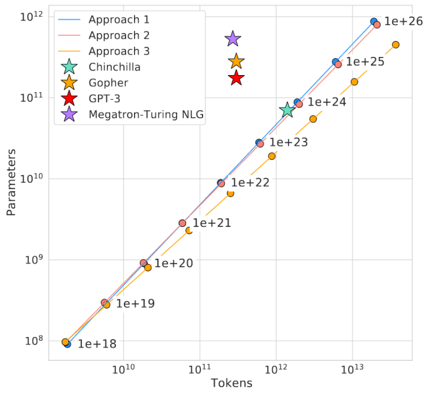


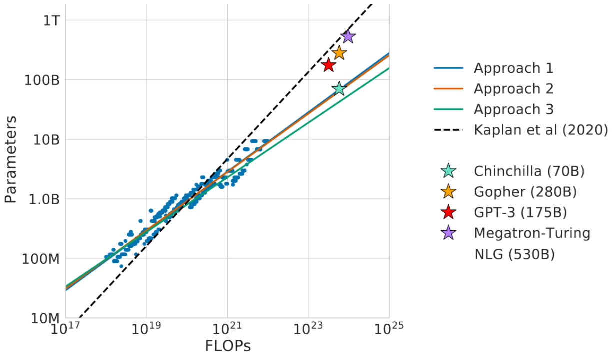
On the left you see curves where optimal loss $L(N,D)$ is constant. The leftmost point of each curve is the lowest FLOPs budget allowing for such loss. Drawing a line through these points, we estimate what is the optimal model size for each FLOPs budget.

On the right you see again model size vs loss. Each dashed curve shows theoretical values of $L(N,D)$ with changing N and fixed FLOPs budget. Again, each curve has its minimum showing the optimal model size given FLOPs budget.

Gathering everything

These three approaches give slightly different results, but on high level they are consistent:





Why Chinchilla paper came as a big surprise for everyone?

Chinchilla wasn't the first paper to evaluate model size / number of tokens trade-off. The previous, and quite an influential one was Scaling Laws for Neural Language Models (Kaplan et al.) by OpenAI that appeared as early as 2020.

The result of this paper is the black dashed line on the right plot above. It predicted that you need to scale the number of parameters faster than the number of training tokens, and models like GPT-3 more or less obeyed to it. What Chinchilla did is it showed that you need to scale them both on par. It also allowed to reason that:

- GPT-3 may have been better if it were trained on significantly greater number of tokens.
- With the same training dataset, GPT-3 could have been smaller without loss of quality.

The difference between analysis in Kaplan et al. and Chinchilla was mostly about making learning rate schedule more realistic.

Why do we only account for the number of parameters and not architecture?

A good question indeed. The one important thing that was proved in the paper by Kaplan et al is that LLM's width does not influence much the loss.

Scaling laws applied

The 70-billion parameter Chinchilla model outperformed the 280-billion parameter Gopher model while using a similar compute budget by being trained on four times more data.

5.2 Scaling laws with constrained datasets

Introduced in Scaling Data-Constrained Language Models.

Chinchilla paper assumed that we have an infinite source of (unique) training tokens. This actually aligns with the fact that many LLMs are pre-trained with only one passage through the training dataset. However, it is not a well funded design choice, and in practice we usually want to extract as much value as possible from the dataset that we were able to obtain. So, why wouldn't we train an LLM on several epochs?

Instead of a single token budget D, we'll have two numbers: number of unique tokens U_D and number of repetitions R_D (it's number of epochs - 1). The authors of this paper train models of different size on datasets of varying size with different number of epochs. They use several approaches of analyzing this data to come up with recommendations about the trade off between number of parameters and number of epochs for a given trained set.

Parametric fit

The authors fit a formula for dependency between L, N, U_D and R_D. In Chinchilla paper, it was

$$L(N, D) = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}.$$

In this paper, the authors suggest using *effective data size* D' and *effective number of parameters* N' instead of simple D and N. Namely:

- We assume that the value of data decreases exponentially with each subsequent epoch:

  $$D' = U_D + (1-\delta)U_D + \ldots + (1-\delta)^{R_D} U_D,$$

  where δ is a learned constant. Denoting $R_D^* = \frac{1}{\delta}$ and making approximations with Taylor formula, we get

  $$D' = U_D + U_D R_D^* \left(1 - e^{-\frac{R_D}{R_D^*}}\right)$$

- The author argue that on later epochs training also offers diminishing returns per parameter suggesting the similar formula for effective number of parameters:

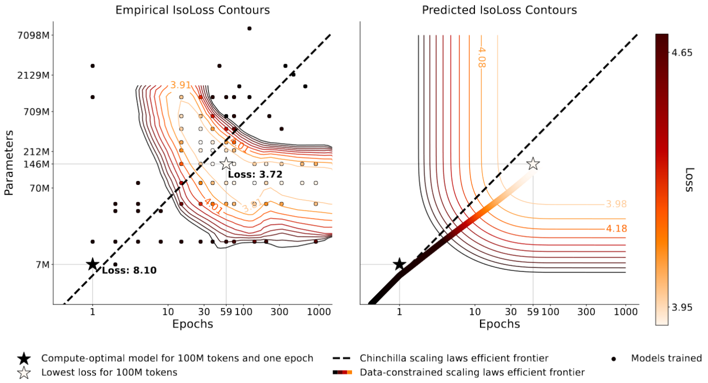
  $$N' = N + N R_N^* \left(1 - e^{-\frac{R_D}{R_N^*}}\right)$$

Using the experiment data, the authors fitted all the parameters. I won't write the whole formula, just note that

$$\alpha = \beta = 0.35, \quad R_N^* = 5.3, \quad R_D^* = 15.4$$

It's curious that α and β are the same.

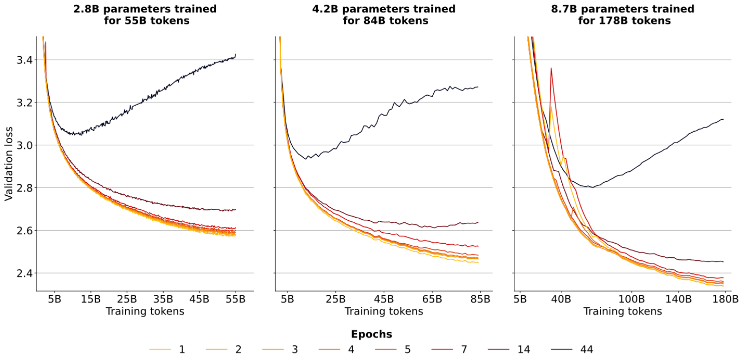
Approach 1. Fixed amount of unique data



On this picture you see the level curves $L = const$, empirical on the left and theoretic (from the formula above) on the right. Unique data size if fixed at 100M tokens. Chinchilla paper law (black dash line) is given just for comparison, it shows what model size would be optimal if we had new data on each epoch.

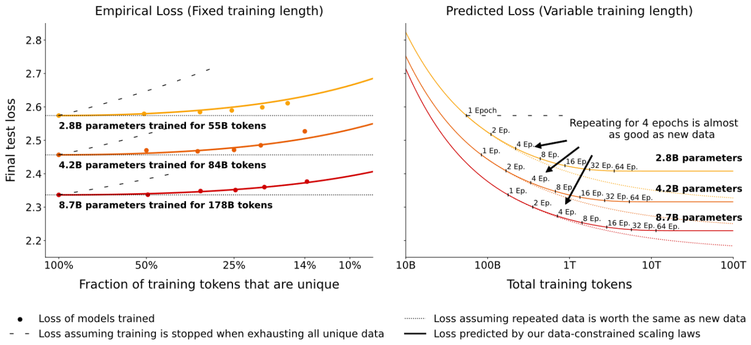
The thick curve on the right goes through minimal FLOPs budget points on each of the $L = const$ curves. It shows what are the optimal models size and number of epochs for getting particular loss value if we train on 100M token unique data. You can see that the thick curve deviates from the Chinchilla law which means that if our dataset is constrained, we have to train for more epochs and choose a model with less parameters.

Approach 2. Fixed Total Tokens



On this plot you see the effect of number of epochs. Please be careful: "Trained on 55B tokens" means that 55B is the sum over all epochs. So, black curves showcase situation when we have many (44) epochs and thus not so many unique tokens at each epoch. Non-surprisingly, with few unique data we're not able to get good quality and even make everything worse after too many epochs.

Predicted loss dynamics during training



This is probably the most important plot in the paper. On the right you see:

- The learning curves that are predicted by the $L(N, D)$ formula above for training on repeated data (solid),
- The learning curves that are predicted by the $L(N, D)$ formula above for training on an unlimited pool of unique data (dotted).

The central observation is: **while training over the first 4 epochs the loss descends almost at the same rate as if we trained on the same amount of unique data, although further epochs are far less effective**. It's good news, because getting unique data is always a challenge.