Survey Scaling Law and Efficiency

Presentation By Aidan, Afsara, Rituparna and Henry

Roadmap

1. Intro and Background, based on Efficient Large Language Models: A Survey, presented by Aidan
2. Scaling Laws for Neural Language Models, presented by Henry
3. LIMA: Less Is More for Alignment, presented by Rituparna Datta
4. The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits, presented by Afsara

Intro, Background, Survey

Based on the paper Efficient Large Language Models: A Survey, https://arxiv.org/abs/2312.03863,
Presented by Aidan Hesselroth (ash2taf)

In the modern era of AI, and specifically with the popularity of LLMs, the resource and time demands of models keep increasing, often with little regard to efficiency. What measures are being taken to improve performance, and how do current models stack up?

Background I

- Ever growing parameter counts
- Better performance on larger, slower models
- Some groups abandon efficiency for better accuracy/reasoning/etc

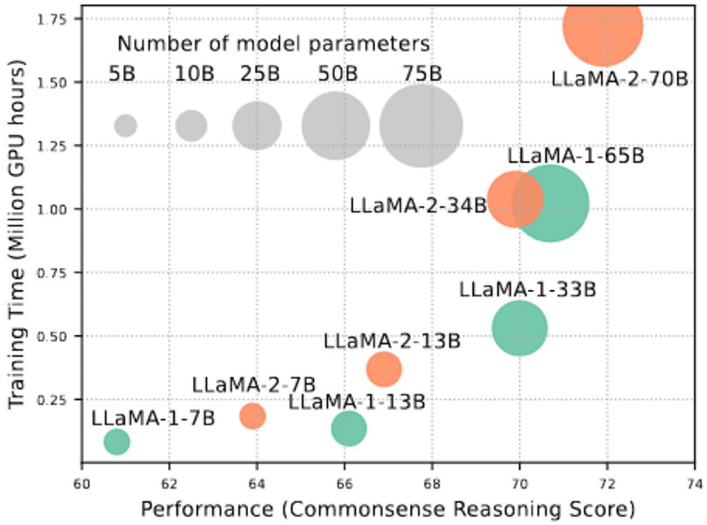


Figure 1: Illustration of model performance and model training time in GPU hours of LLaMA models at different scales. The reported performance is the average score of several commonsense reasoning benchmarks. The training time is based on Nvidia A100 80GB GPU. The size of each circle corresponds to the number of model parameters. The original data can be found in Touvron et al. (2023a;b).

Background II

- Obviously, there are some models more focused on efficiency than others
- Look at Mistral 7B, LLama-2-7B, and LLaMa-1-33B

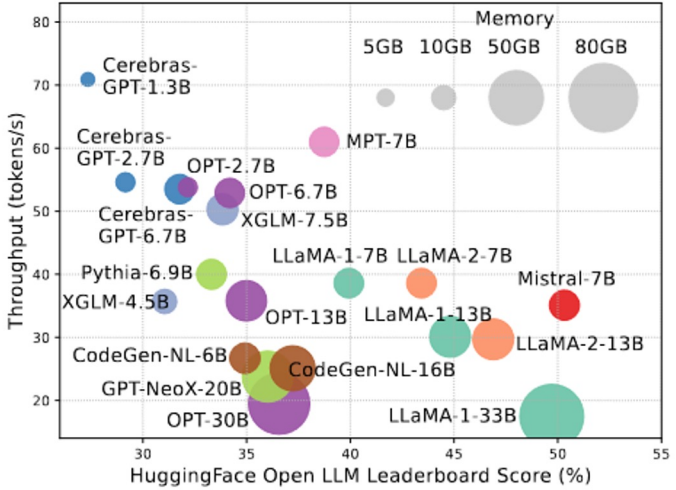


Figure 2: Performance score *vs.* inference throughput for various LLMs. The throughputs are measured on Nvidia A100 80GB GPU with 16-bit floating point quantization. The size of each circle corresponds to the memory footprint (in Gigabytes) of each model when running with batch size of 1, prompt size of 256 and generating 1000 tokens. The original data can be found in Ilyas Moutawwakil (2023).

Efficiency Taxonomy

3 Categories of efficiency techniques:

1. **Model-Centric Methods:** research directions related to model compression, efficient pre-training, efficient fine-tuning, efficient inference, and efficient architecture design
2. **Data-Centric Methods:** research directions related to data selection and prompt engineering
3. **LLM Frameworks:** existing frameworks specifically designed for efficient LLMs, addressing their unique features, underlying libraries, and specialization

Taxonomy Diagram

- Don’t let the diagram fool you, while there is less variety in some, all 3 areas are relatively ‘hot’
- Specifically focused on LARGE Language models, unlike previous surveys
- Obviously out of scope to go over in detail, so we’ll skim

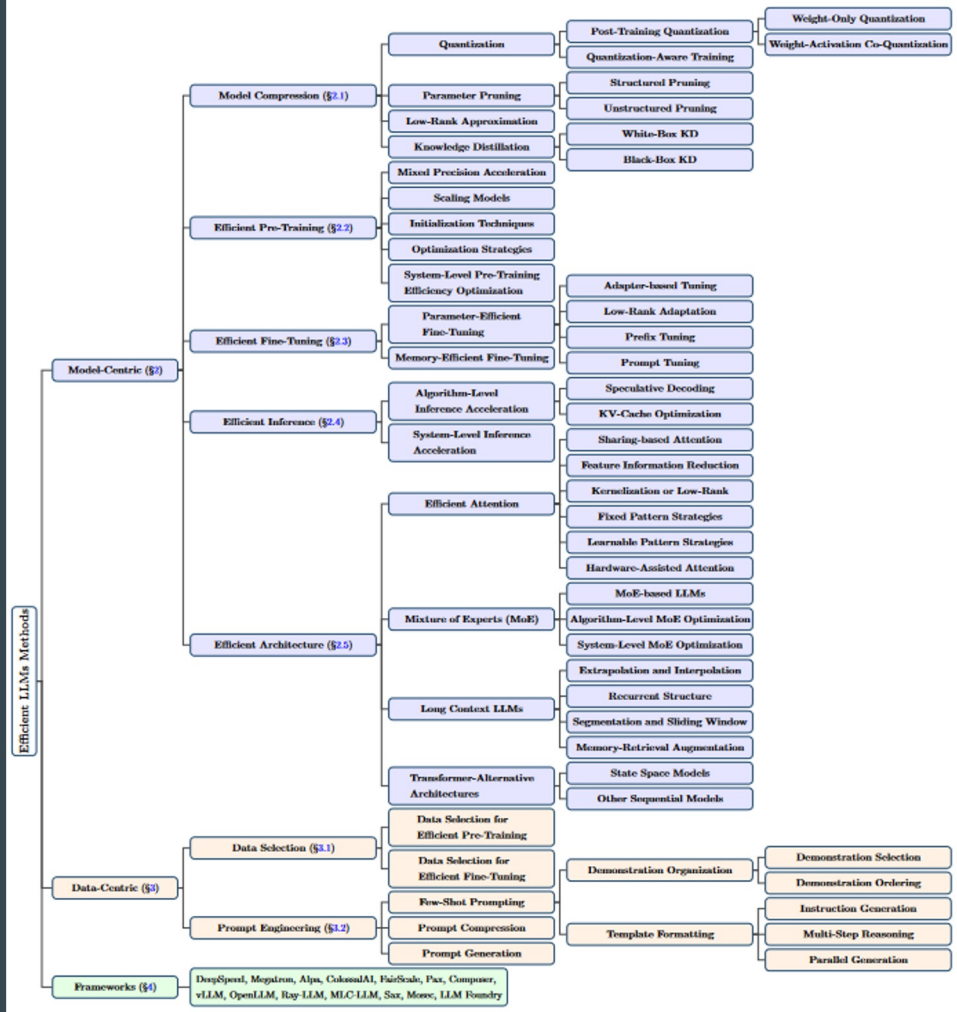


Figure 3: Taxonomy of efficient large language models (LLMs) literature.

Model Compression

"As summarized in Figure 4, model compression techniques for LLMs can be grouped into four categories: quantization, parameter pruning, low-rank approximation, and knowledge distillation"

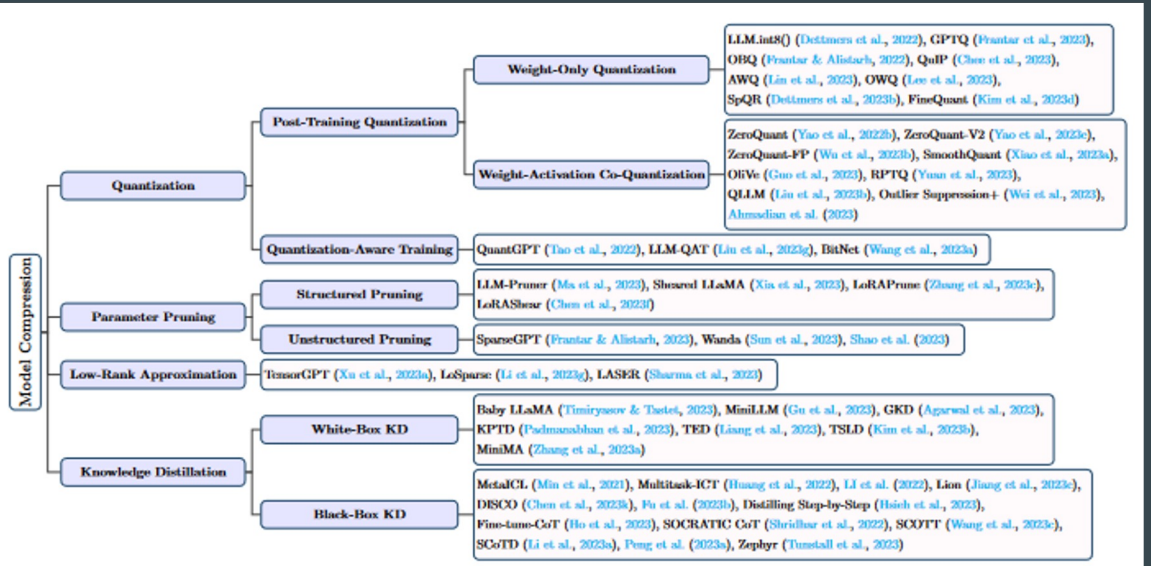


Figure 4: Summary of model compression techniques for LLMs.

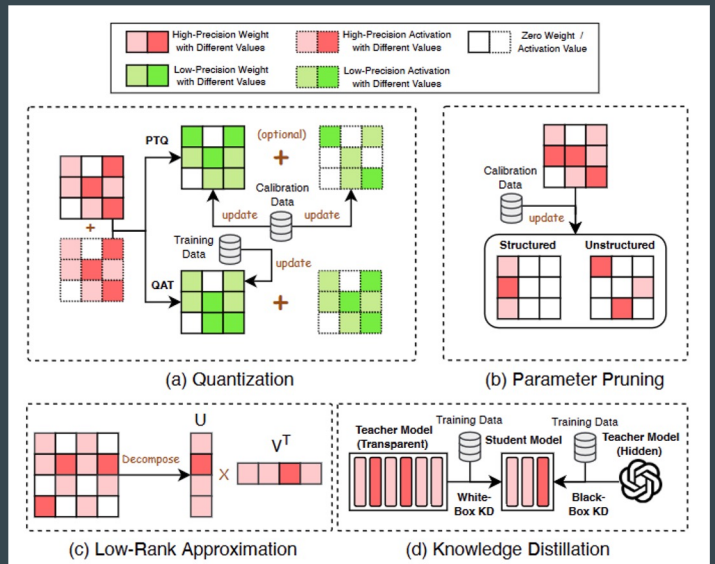


Figure 5: Illustrations of model compression techniques for LLMs.

Efficient Pre-Training

"As shown in Table 1, pre-training LLMs incurs high costs. Efficient pre-training aims to enhance the efficiency and reduce the cost of the LLM pre-training process. As summarized in Figure 6, efficient pre-training techniques can be grouped into four categories: mixed precision acceleration, scaling models, initialization techniques, and optimization strategies"

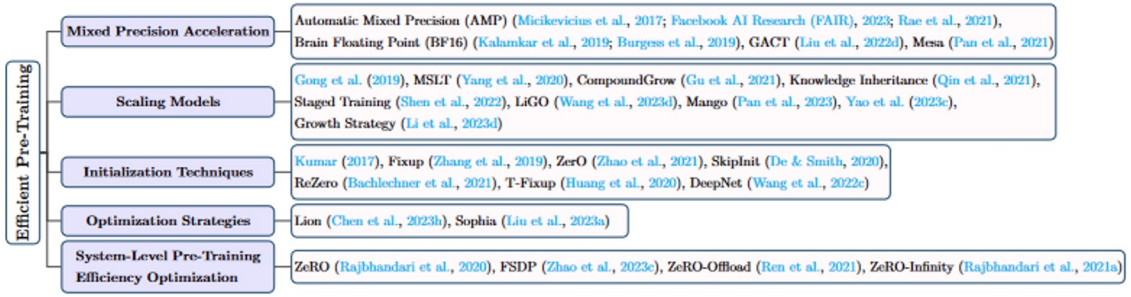


Figure 6: Summary of efficient pre-training techniques for LLMs.

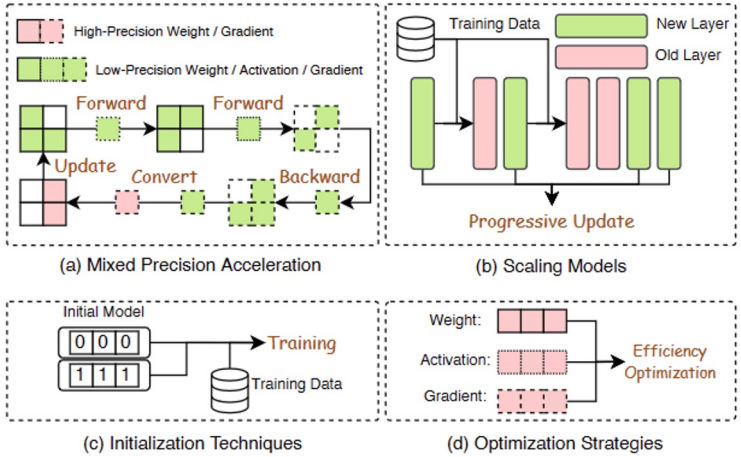


Figure 7: Illustrations of efficient pre-training techniques for LLM.

Table 1: Pre-training costs of representative LLMs.

Model	Parameter Size	Data Scale	GPUs Cost	Training Time
GPT-3 (Brown et al., 2020)	175B	300B tokens	-	-
GPT-NeoX-20B (Black et al., 2022)	20B	825GB corpus	96 A100-40G	-
OPT (Zhang et al., 2022a)	175B	180B tokens	992 A100-80G	-
BLOOM (Scao et al., 2022)	176B	366B tokens	384 A100-80G	105 days
GLM (Zeng et al., 2022)	130B	400B tokens	786 A100-40G	60 days
LLaMA (Touvron et al., 2023a)	65B	1.4T tokens	2048 A100-80G	21 days
LLaMA-2 (Touvron et al., 2023b)	70B	2T tokens	A100-80G	71,680 GPU days
Gopher (Rae et al., 2021)	280B	300B tokens	1024 A100	13.4 days
LaMDA (Thoppilan et al., 2022)	137B	768B tokens	1024 TPU-v3	57.7 days
GLaM (Du et al., 2022)	1200B	280B tokens	1024 TPU-v4	574 hours
PanGu-α (Zeng et al., 2021)	13B	1.1TB corpus	2048 Ascend 910	-
PanGu-$\sum$ (Ren et al., 2023b)	1085B	329B tokens	512 Ascend 910	100 days
PaLM (Chowdhery et al., 2022)	540B	780B tokens	6144 TPU-v4	-
PaLM-2 (Anil et al., 2023)	-	3.6T tokens	TPUv4	-
WeLM (Su et al., 2022b)	10B	300B tokens	128 A100-40G	24 days
Flan-PaLM (Chung et al., 2022)	540B	-	512 TPU-v4	37 hours
AlexaTM (Soltan et al., 2022)	20B	1.3 tokens	128 A100	120 days
Codegeex (Zheng et al., 2023)	13B	850 tokens	1536 Ascend 910	60 days
MPT-7B (Team, 2023)	7B	1T tokens	-	-

Fine-tuning

“Efficient fine-tuning aims to enhance the efficiency of the fine-tuning process for LLMs. As shown in Figure 8, efficient fine-tuning methods can be grouped into parameter-efficient fine-tuning (PEFT), and memory- efficient fine-tuning (MEFT).

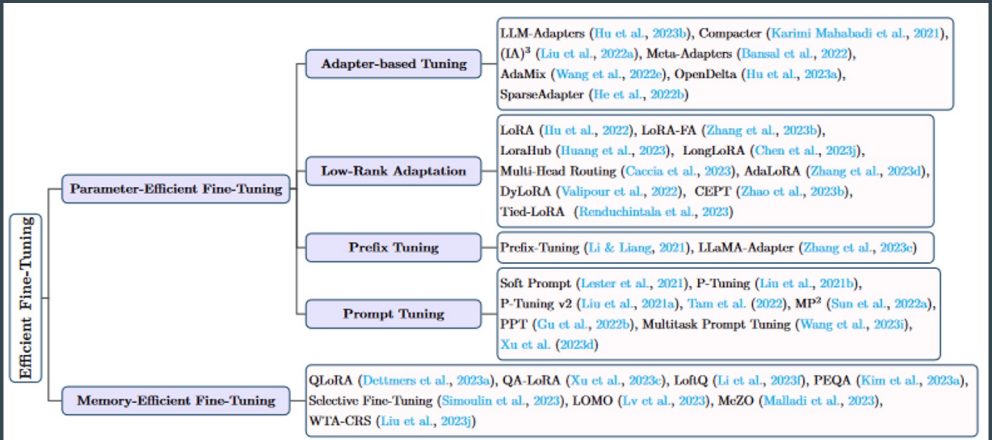


Figure 8: Summary of efficient fine-tuning methods for LLMs.

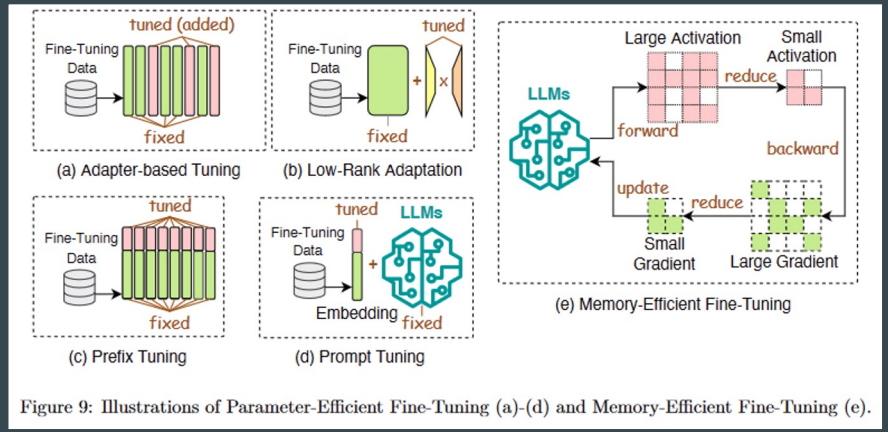


Figure 9: Illustrations of Parameter-Efficient Fine-Tuning (a)-(d) and Memory-Efficient Fine-Tuning (e).

Efficient Inference

"Efficient inference aims to enhance the efficiency of the inference process for LLMs. As summarized in Figure 10, efficient inference techniques can be grouped into techniques at the algorithm level and system level."

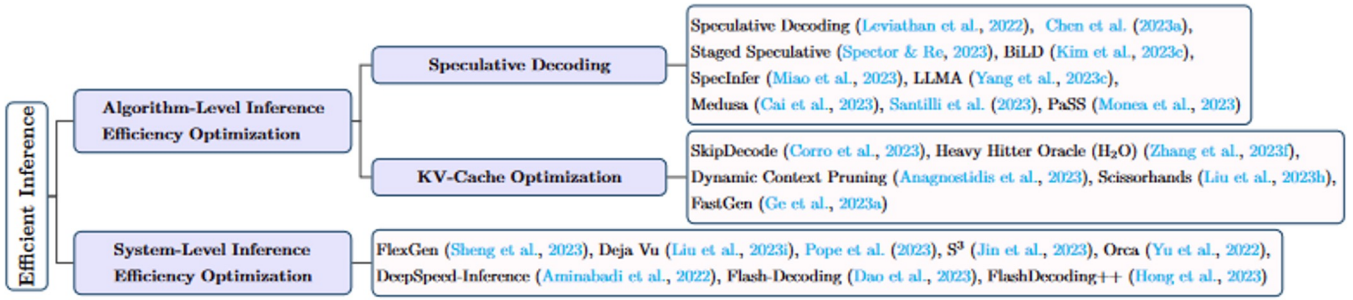


Figure 10: Summary of efficient inference techniques for LLMs.

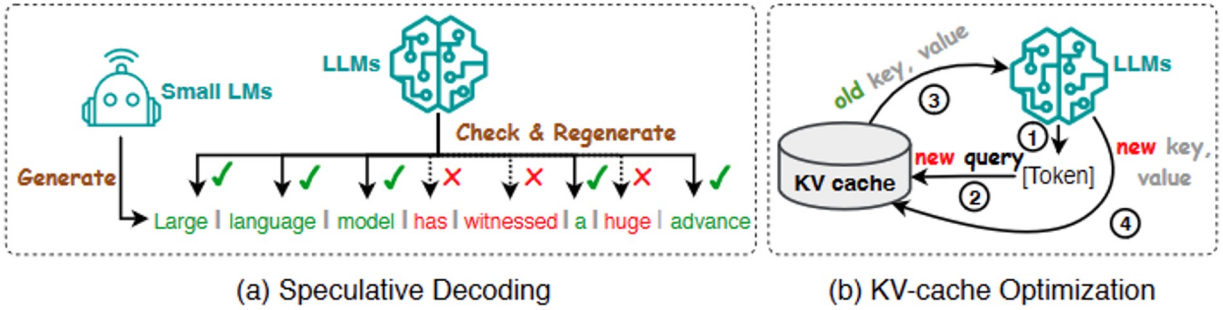


Figure 11: Illustrations of algorithm-level efficiency optimization techniques for LLM inference.

Efficient Architecture

"Efficient architecture design for LLMs refers to the strategic optimization of model architecture and computational processes to enhance performance and scalability while minimizing resource consumption. Figure 12 summarizes efficient architecture designs for LLMs"

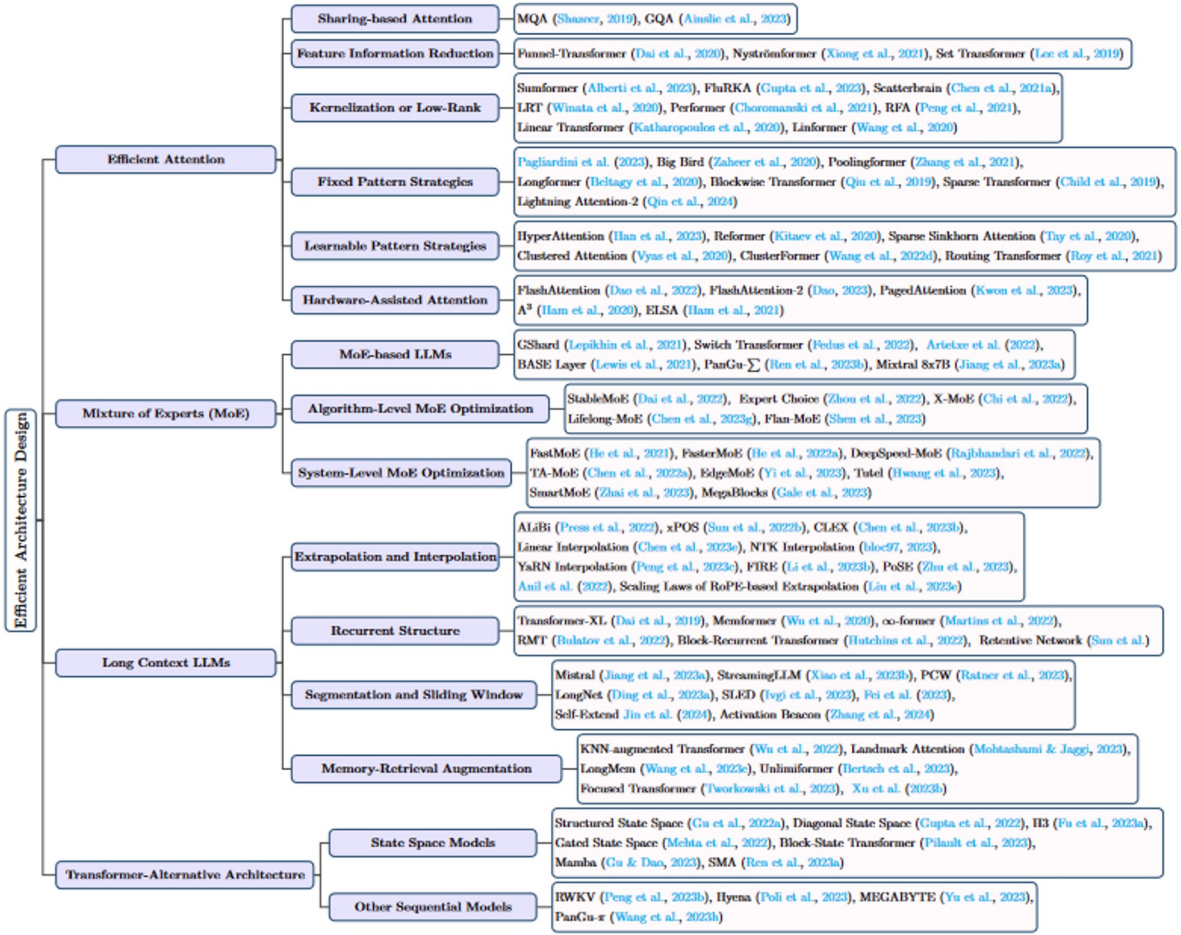


Figure 12: Summary of efficient architecture designs for LLMs.

Data Centric

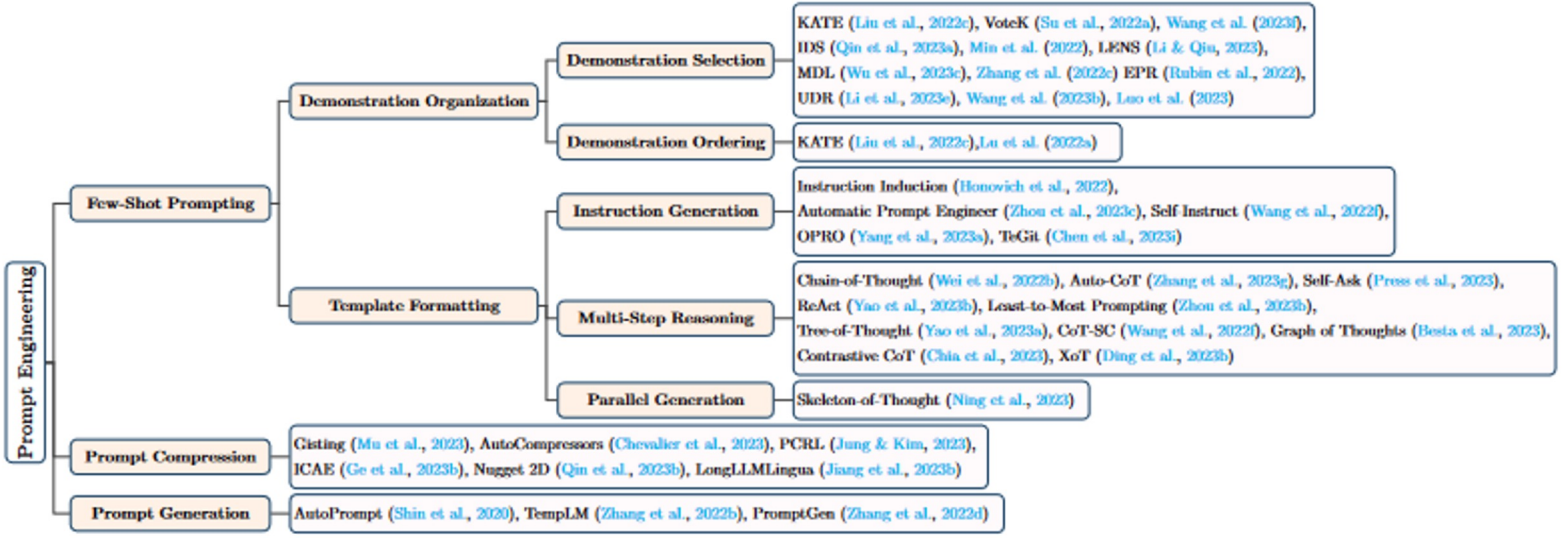


Figure 17: Summary of prompt engineering techniques for LLMs.

Data Centric: Few Shot Prompting

By training to work with few shot scenarios, further training costs avoided and increases speed of adaptation

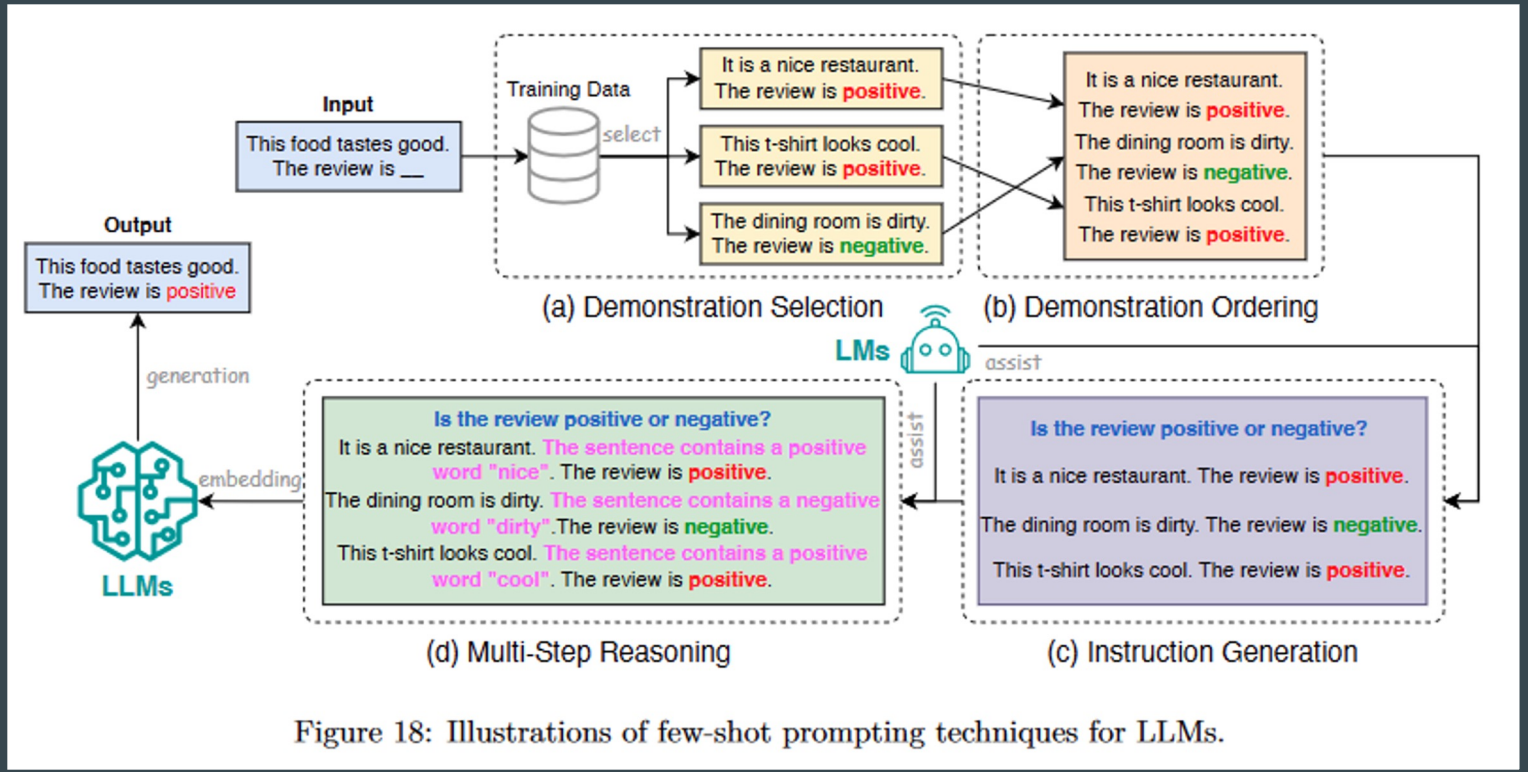


Figure 18: Illustrations of few-shot prompting techniques for LLMs.

Data Centric: Prompt Compression and Generation

Prompt compression via condensing inputs or compact prompt representation allows for denser information, reducing size in memory, time to query, etc. Prompt Generation automatically creates optimized prompts to improve performance even with unskilled users

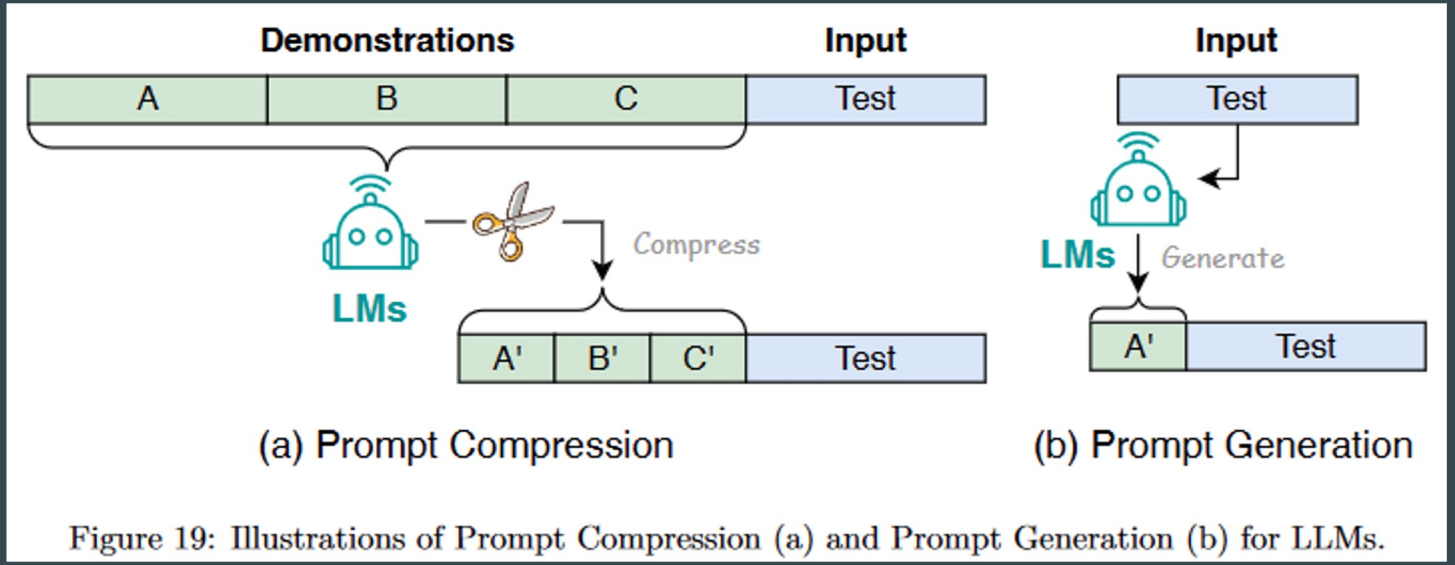


Figure 19: Illustrations of Prompt Compression (a) and Prompt Generation (b) for LLMs.

Frameworks

More details on individual frameworks can be found in section 4 of the paper, recommended to check there if you want a better description

Framework	Training	Fine-Tuning	Inference	Features
DeepSpeed	✓	✓	✓	Data Parallelism, Model Parallelism, Pipeline Parallelism, Prompt Batching, Quantisation, Kernel Optimizations, Compression, Mixture of Experts.
Megatron	✓	✓	✓	Data Parallelism, Model Parallelism, Pipeline Parallelism, Prompt Batching, Automatic Mixed precision, Selective activation Recomputation
Alpa	✓	✓	✓	Data Parallelism, Model Parallelism, Pipeline Parallelism, Operator Parallelism, Automated Model-Parallel Training, Prompt Batching
Colossal AI	✓	✓	✓	Data Parallelism, Model Parallelism, Pipeline Parallelism, Mixed Precision Training, Gradient accumulation, heterogeneous Distributed Training, Prompt Batching, Quantization
FairScale	✓	✓	✓	Data Parallelism, Model Parallelism, Pipeline Parallelism, Activation Checkpointing, Model Offloading, Model scaling, Adascale Optimization
Pax	✓	✓	✓	Data Parallelism, Model Parallelism, Kernel Optimization
Composer	✓	✓	✓	Fully Sharded Data Parallelism, Elastic sharded checkpointing, Flash Attention
vLLM	✗	✗	✓	Data Parallelism, Model Parallelism, Tensor Paralelism, Efficient management via PagedAttention, Optimized CUDA kernels, Dynamic Batching, Quantization
OpenLLM	✗	✓	✓	Distributed Finetuning and Inference, Integration with BentoML, LangChain, and Transformers Agents, Prometheus Metrics, Token Streaming
Ray LLM	✗	✗	✓	Distributed Inference, Integration with Alpa, Prompt Batching, Quantization, Prometheus Metrics
MLC LLM	✗	✗	✓	Distributed Inference, Compiler Acceleration, Prompt Batching, Quantization
Sax	✗	✗	✓	Distribute Inference, Serves PaxML, JAX, and PyTorch models, Slice Serving, Prometheus Metrics
Mosec	✗	✗	✓	Distribute Inference, Dynamic Batching, Rust-based Task Coordinator, Prometheus Metrics
LLM Foundry	✗	✗	✓	Distribute Inference, Dynamic Batching, Prompt Batching

Taxonomy Diagram (Again)

- Returning to the full tree, you can see the 3 areas again
- All of them are deep, but model centric is easily the broadest
- This paper is a great way to find relevant papers for any of these areas that sounded interesting to you!

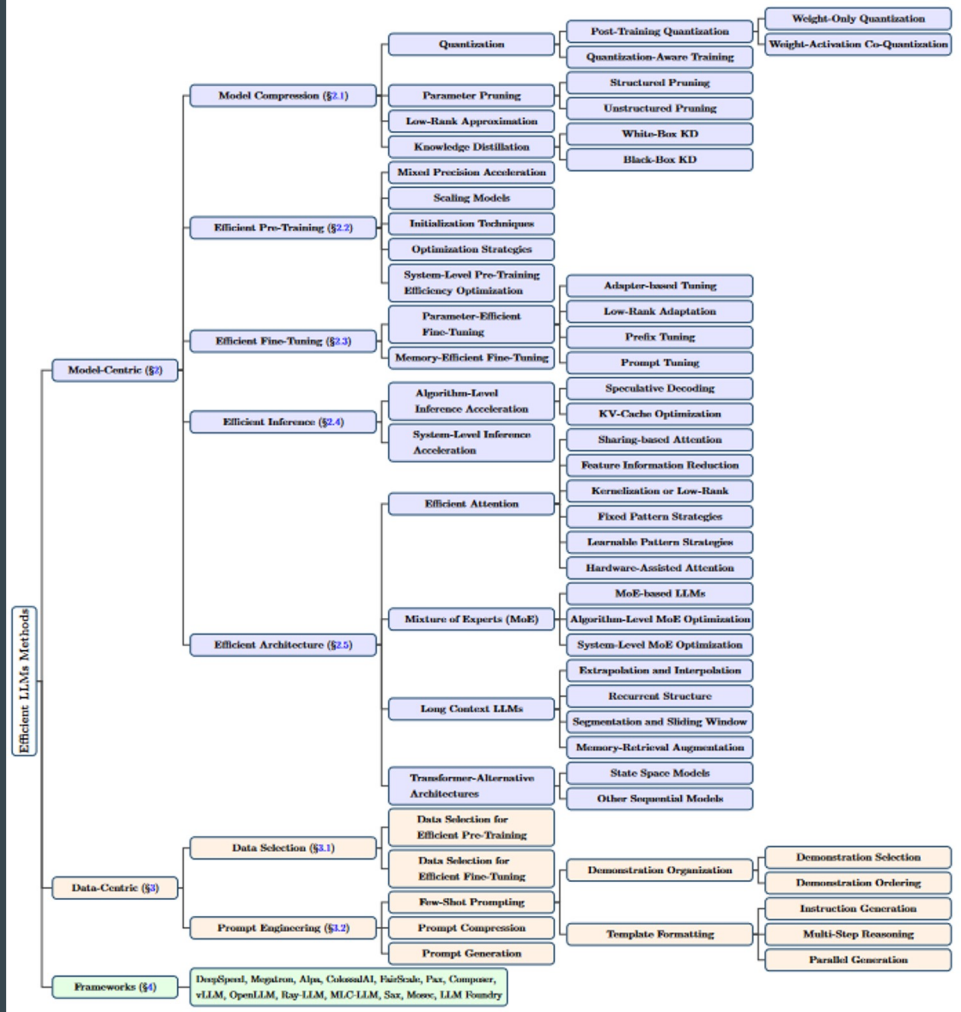


Figure 3: Taxonomy of efficient large language models (LLMs) literature.

Scaling Laws for Neural Language Models

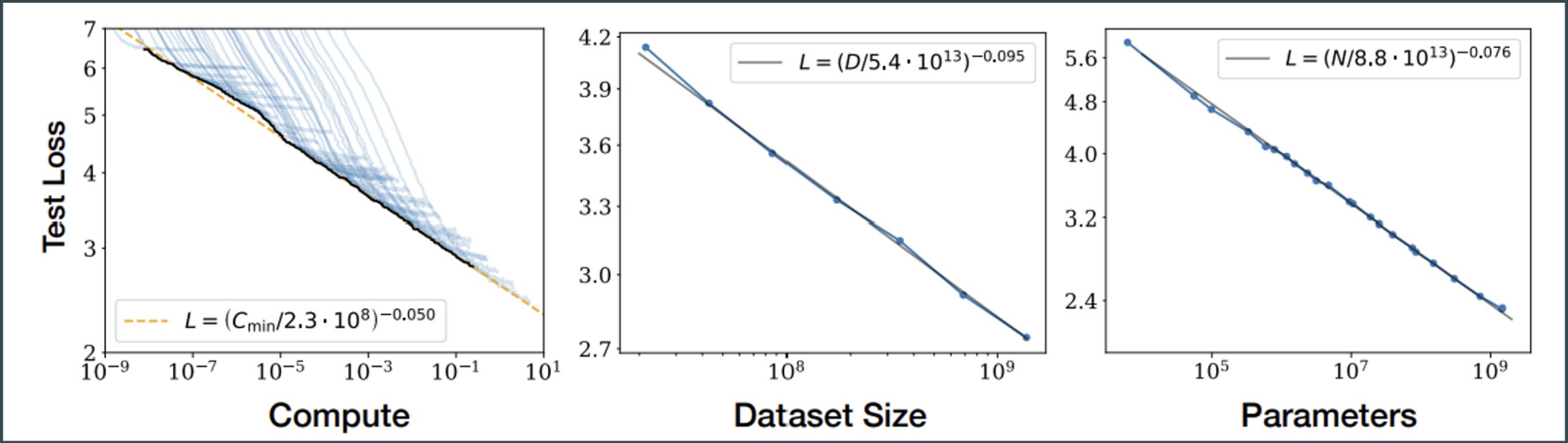
Presented by: Henry Radzikowski

Language serves as a natural domain for AI research, facilitating reasoning tasks and offering abundant textual data for unsupervised learning.

Empirical Results and Basic Power Laws

- To characterize language model scaling we train a wide variety of models, varying a number of factors including:
  - Model size (ranging in size from 786 to 1.5 billion non-embedding parameters)
  - Data size (ranging from 22 million to 23 billion tokens)
  - Shape (including depth, width, attention heads, and feed-forward dimension)
  - Context length (1024 for most runs)
  - Batch size (2^19 for most runs, sometimes varied to measure critical batch size)

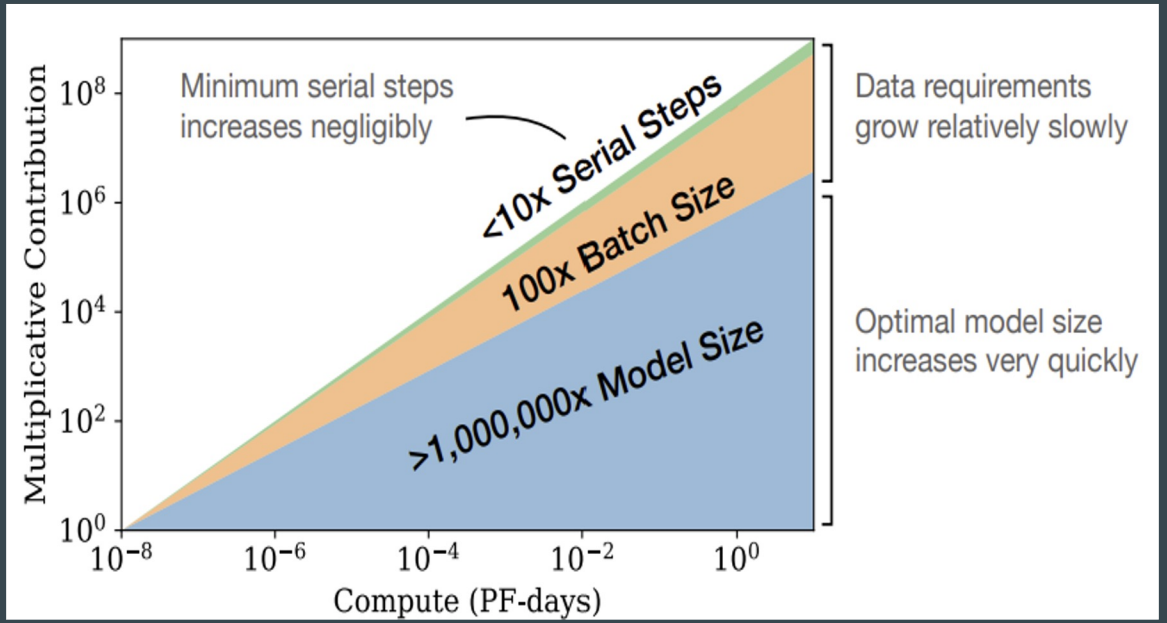
Summary



Language modeling performance improves smoothly as we increase model size, dataset size, and amount of computing power used for training. For optimal performance, all three must be scaled up together.

Optimizing Compute Efficiency in Training AI Models

With increased computational resources, optimizing training efficiency involves allocating the majority towards lager model sizes, with inversely smaller increase in data.



Notation

- L – the cross entropy loss in nats. Typically it will be averaged over the tokens in a context, but in some cases we report the loss for specific tokens within the context.
- N – the number of model parameters, *excluding all vocabulary and positional embeddings*
- $C \approx 6NBS$ – an estimate of the total non-embedding training compute, where B is the batch size, and S is the number of training steps (ie parameter updates). We quote numerical values in PF-days, where one PF-day $= 10^{15} \times 24 \times 3600 = 8.64 \times 10^{19}$ floating point operations.
- D – the dataset size in tokens
- B_{crit} – the critical batch size [MKAT18], defined and discussed in Section 5.1. Training at the critical batch size provides a roughly optimal compromise between time and compute efficiency.
- C_{min} – an estimate of the minimum amount of non-embedding compute to reach a given value of the loss. This is the training compute that would be used if the model were trained at a batch size much less than the critical batch size.
- S_{min} – an estimate of the minimal number of training steps needed to reach a given value of the loss. This is also the number of training steps that would be used if the model were trained at a batch size much greater than the critical batch size.
- α_X – power-law exponents for the scaling of the loss as $L(X) \propto 1/X^{\alpha_X}$ where X can be any of $N, D, C, S, B, C^{\text{min}}$.

Summary of Scaling Laws

1. For models with a limited number of parameters, trained to convergence on sufficiently large datasets:

$$L(N) = (N_c/N)^{\alpha_N}\ ;\ \ \alpha_N \sim 0.076, \quad N_c \sim 8.8 \times 10^{13} \text{ (non-embedding parameters)} \tag{1.1}$$

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

$$L(D) = (D_c/D)^{\alpha_D}\ ;\ \ \alpha_D \sim 0.095, \quad D_c \sim 5.4 \times 10^{13} \text{ (tokens)} \tag{1.2}$$

3. When training with a limited amount of compute, a sufficiently large dataset, an optimally-sized model, and a sufficiently small batch size (making optimal[3] use of compute):

$$L(C_{\min}) = \left(C_c^{\min}/C_{\min}\right)^{\alpha_C^{\min}}\ ;\ \ \alpha_C^{\min} \sim 0.050, \quad C_c^{\min} \sim 3.1 \times 10^{8} \text{ (PF-days)} \tag{1.3}$$

Eq 1.1 - predicts the test loss of a transformer model with constrained by N.

Eq 1.2 - predicts the test loss on large models trained on limited data and early stopping.

Eq 1.3 - describes test loss when training with limited amount of compute, large dataset, optimally-sized model, and small batch size.

Model-Data Scaling Relationship in Language Modeling

$$L(N, D) = \left[\left(\frac{N_c}{N} \right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D} \right]^{\alpha_D} \quad (1.5)$$

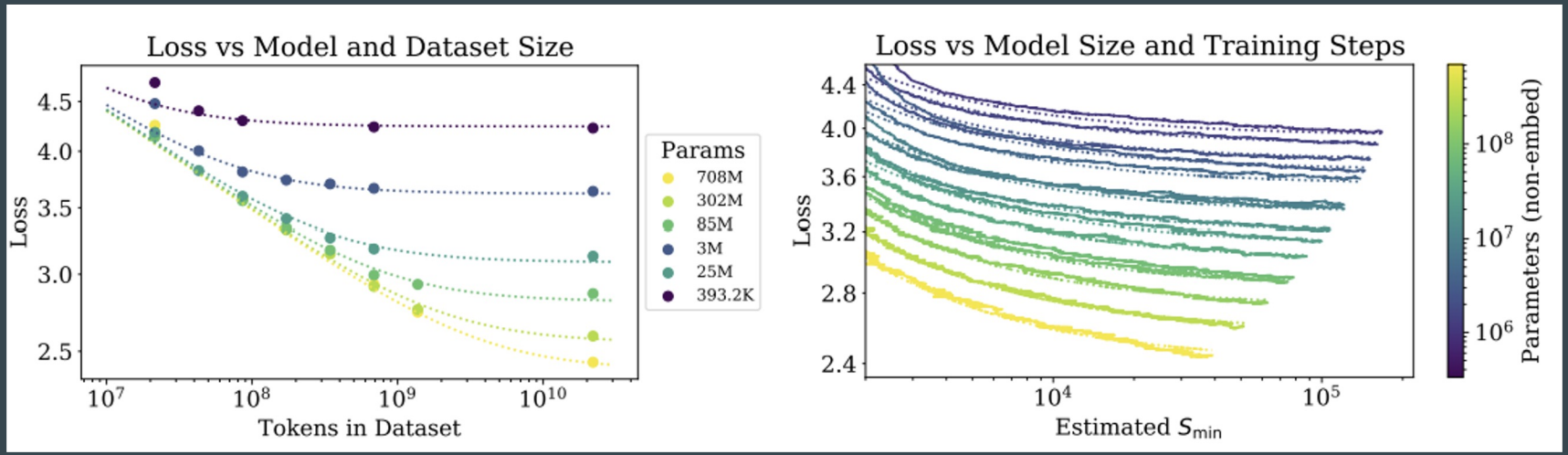
- Eq 1.5 combines the impact of both model size and dataset size on test loss and overfitting.
- Indicates sublinear increase in dataset size relative to model size, crucial for optimizing performance and mitigating overfitting in language modeling size.

Model Training Dynamics: Optimizing Performance (Finite)

$$L(N, S) = \left(\frac{N_c}{N}\right)^{\alpha_N} + \left(\frac{S_c}{S_{\min}(S)}\right)^{\alpha_S} \quad (1.6)$$

- Equation (1.6) characterizes the learning curves of a model during training within a fixed computational budget (C) and a finite number of parameter update steps.
- It illustrates how the test loss (L) is influenced by model size (N), the number of parameter update steps (S), and the minimum possible number of steps (Smin), offering insights into optimizing model training efficiency and resource allocation.

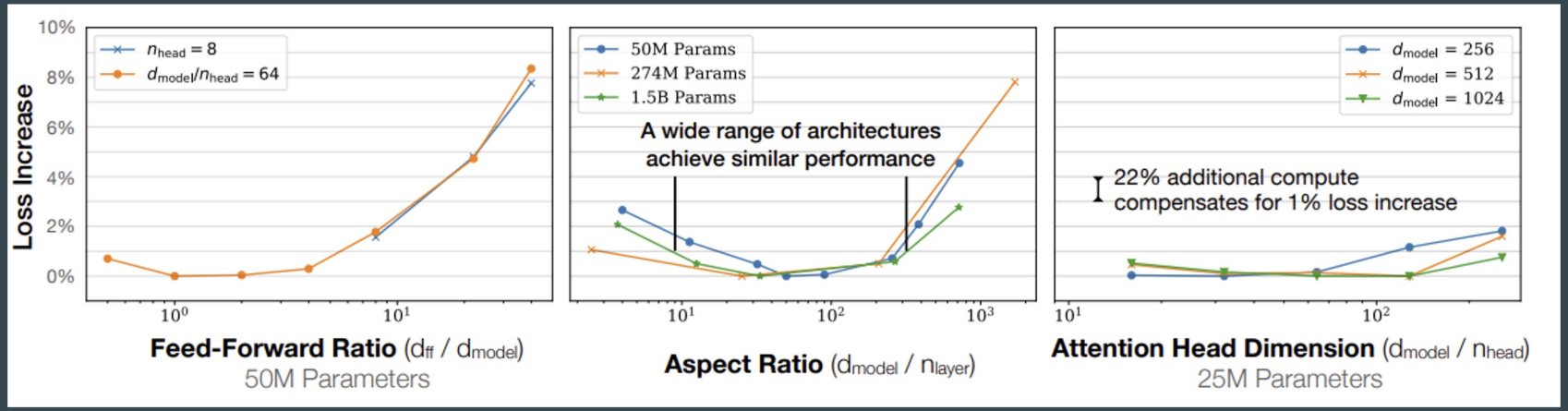
Efficient Language Model Training



Left - The test loss varies predictably with both dataset size and model size. (eq 1.5)

Right - Learning curves for different model sizes can be accurately modeled, where Smin represents the number of steps for a large batch size, holding true for various orders of magnitude. (eq 1.6)

Model Performance



Performance depends very mildly on model shape when the total number of non-embedding parameters N is held fixed. The loss varies only a few percent over a wide range of shapes. Small differences in parameter counts are compensated for by using the fit to L(N) as a baseline. Aspect ratio in particular can vary by a factor of 40 while only slightly impacting performance; an (nlayer, dmodel) = (6, 4288) reaches a loss within 3% of the (48, 1600) models used.

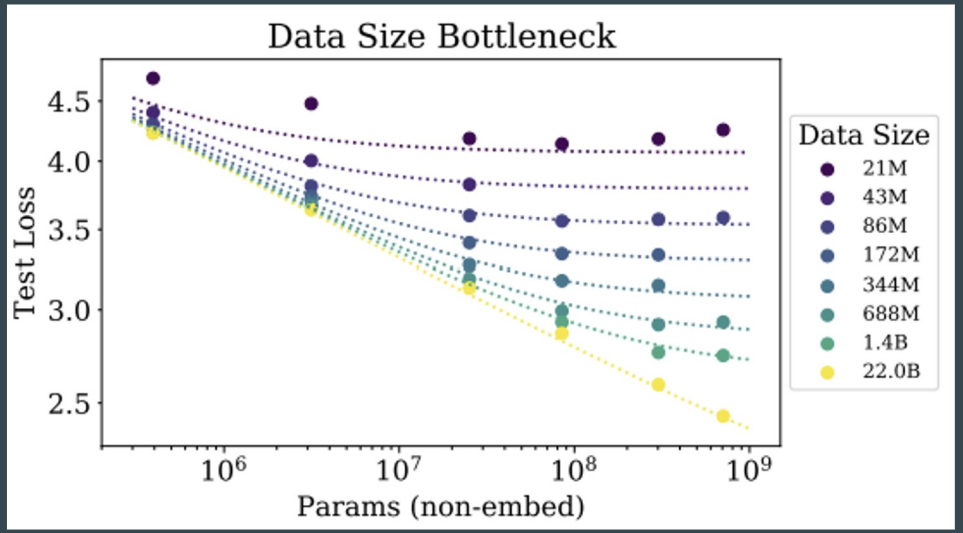
Optimal Loss Parameterization: L(N,D) Equation

$$L(N, D) = \left[\left(\frac{N_c}{N} \right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D} \right]^{\alpha_D} \quad (4.1)$$

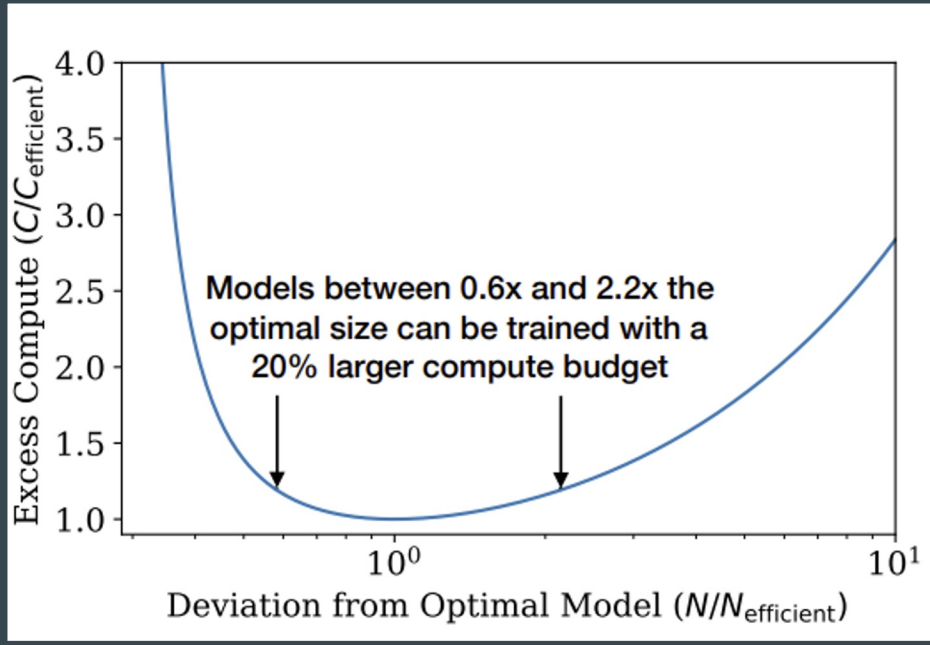
- Equation (4.1) (based on eq 1.5) defines the proposed parameterization for the test loss (L) as a function of model size (N) and dataset size (D).
- It adheres to three key principles: accommodating rescaling due to changes in vocabulary size or tokenization, ensuring convergence to individual losses L(N) and L(D) as N or D approach infinity, and maintaining analyticity at infinite dataset size to support series expansion.

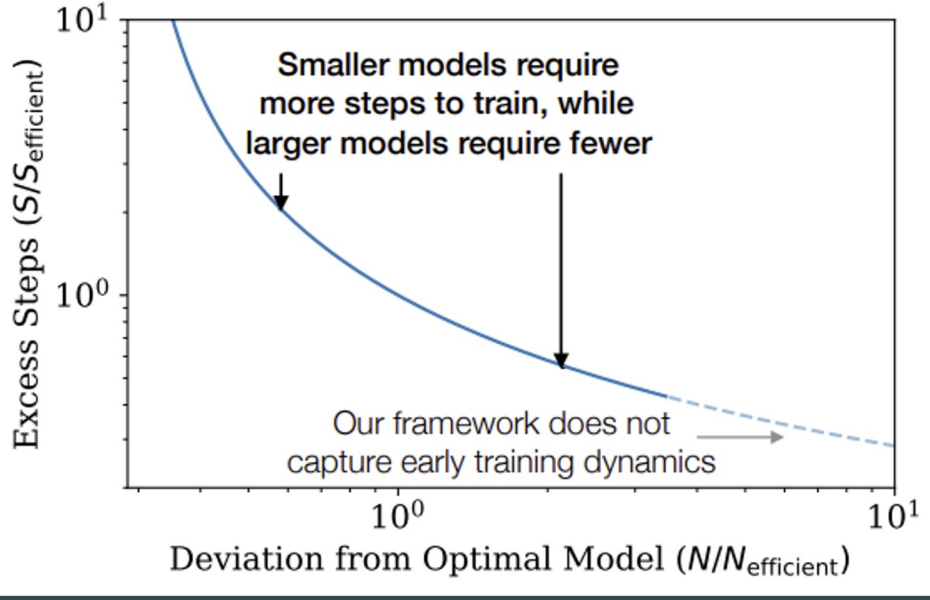
Predictable Dependency of Test Loss on Model and Dataset Size

- Equation 4.1 shows the relationship between early-stopped test loss and both dataset size and model size.



Optimal Allocation of the Compute Budget





LIMA: Less Is More for Alignment

Presented by: Rituparna Datta

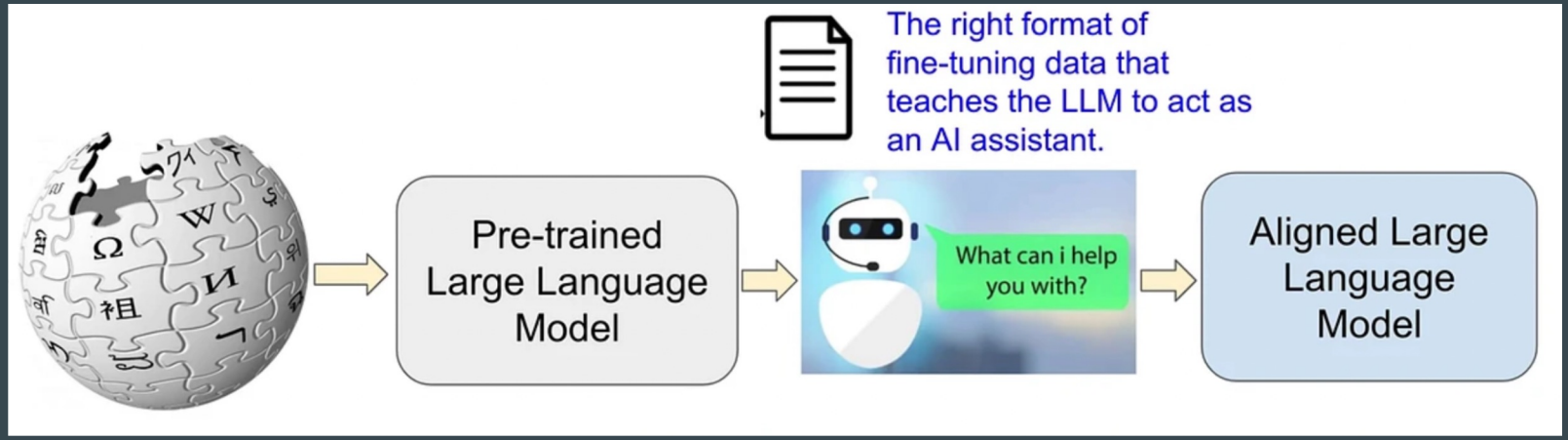
LIMA demonstrates strong performance even with minimal fine-tuning, suggesting that the bulk of their knowledge is acquired during unsupervised pre training rather than large-scale instruction tuning.

Research Questions

1. Do we need large amount of annotated data to train a competent chatbot?
2. What are the critical axes when creating the annotated data?
3. How well can a model trained with a small number of annotated data generalizes to new tasks?

Superficial Alignment Hypothesis

- A model's knowledge and capabilities are learnt almost entirely during pre-training
- Alignment teaches it which subdistribution of format should be used while interacting with users



One could sufficiently tune a pre-trained language model with a rather small set of examples

Alignment Data

Source	#Examples	Avg Input Len.	Avg Output Len.
Training			
Stack Exchange (STEM)	200	117	523
Stack Exchange (Other)	200	119	530
wikiHow	200	12	1,811
Pushshift r/WritingPrompts	150	34	274
Natural Instructions	50	236	92
Paper Authors (Group A)	200	40	334
Dev			
Paper Authors (Group A)	50	36	N/A
Test			
Pushshift r/AskReddit	70	30	N/A
Paper Authors (Group B)	230	31	N/A

Quality and Diversity are the keys!

Quality Control

- For public data: remove artifacts and select data with higher user ratings
- For in house authored data(200): Set a uniform tone and format

Diversity Control

- For public data: stratified sampling to increase **domain diversity**
- For in house authored data(200): come up with different scenarios to increase **task/scenario diversity**

Training Setup & Methodology

Training

- LLaMa 65B [Touvron et al., 2023]
- fine-tune on 1,000-example
- Standard Fine Tuning params:
  - Finetune 15 epochs with AdamW
  - $\beta_1 = 0.9$ $\beta_2 = 0.95$; weight decay= 0.1
  - Batch size = 32
- residual dropout: Ouyang et al. [2022] and apply dropout over residual connections

Annotation Methodology

- assign one point if both annotators agreed
- half a point if either annotator agreed, (but not both) labeled a tie,
- zero points otherwise.
- measure agreement over a shared set of 50 annotation examples (single prompt, two model responses – all chosen randomly), comparing author, crowd, and GPT-4 annotations.

Experiment Setup

To compare LIMA to other Baselines

- generate a single response for each test prompt.
- ask crowd workers to compare LIMA outputs to each of the baselines
- repeat this experiment, replacing human crowd workers with GPT-4

Baselines:

- **Alpaca 65B [Taori et al., 2023]** – finetune LLaMa 65B on the 52,000 examples in the Alpaca training set [Taori et al., 2023];
- **OpenAI's DaVinci003**, a large language model tuned with reinforcement learning from human feedback (RLHF) [Ouyang et al., 2022];
- **Google's Bard**, based on PaLM [Chowdhery et al., 2022]
- **Anthropic's Claude,** 4 a 52B parameter model trained with reinforcement learning from AI
- **OpenAI's GPT-4** [OpenAI, 2023], a large language model trained with RLHF, which is currently considered the state of the art

Lima performs pretty well with 1000 examples

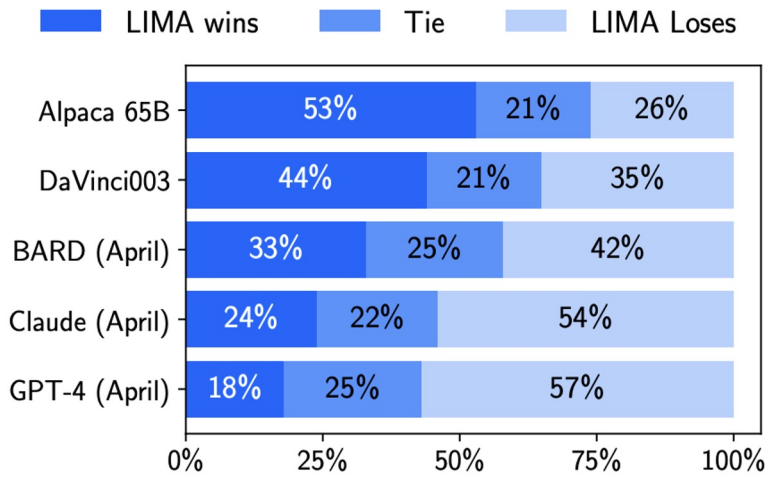


Figure 1: Human preference evaluation, comparing LIMA to 5 different baselines across 300 test prompts.

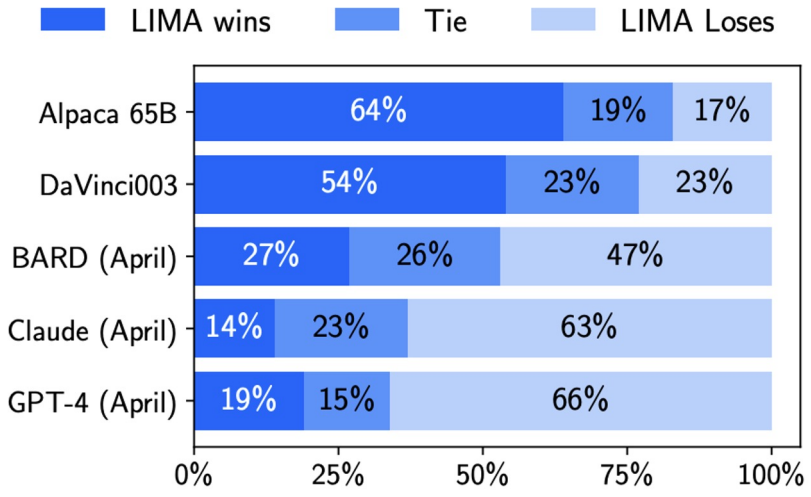


Figure 2: Preference evaluation using GPT-4 as the annotator, given the same instructions provided to humans.

Why is Less More? Ablations on Data Diversity, Quality, and Quantity

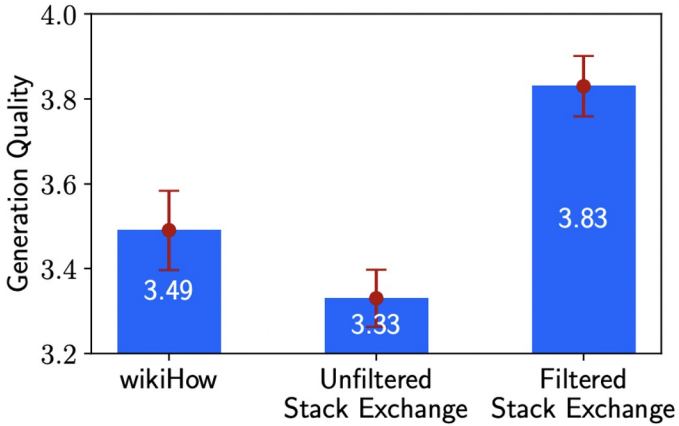


Figure 5: Performance of 7B models trained with 2,000 examples from different sources. **Filtered Stack Exchange** contains diverse prompts and high quality responses; **Unfiltered Stack Exchange** is diverse, but does not have any quality filters; **wikiHow** has high quality responses, but all of its prompts are "how to" questions.

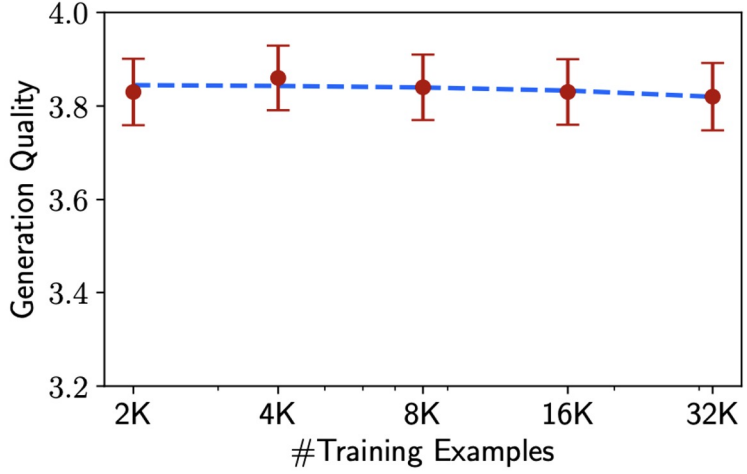


Figure 6: Performance of 7B models trained with exponentially increasing amounts of data, sampled from (quality-filtered) Stack Exchange. Despite an up to 16-fold increase in data size, performance as measured by ChatGPT plateaus.

Task Generalization Capability with a Few Example

Two Showcases:

- 30 multi turn dialogue data improves dialogue capabilities of LIMA
- Adding 6 format constraint examples enables model to generate long form highly structured response following user instructions:
  - Training example : Review a paper from the following aspects: summary, strengths, weaknesses, potentials
  - Test example: Create a marketing plan with the following elements: Marketing goal and objectives, Define Target Audience, Research Marketing Tactics, Plan Marketing Tactics and Develop Your Timeline and Budget

Task Generalization Capability with a Few Example

Here's a marketing plan for your local coffee shop.
Marketing Goals and Objectives

- Increase awareness of our brand among consumers [...]
- Drive foot traffic [...] during slow business hours [...]
- Build relationships with influencers in our community who [...]

Define Target Audience
Our target audiences include:

- Residents of the neighborhood surrounding our coffee shop
- Office workers near our coffee shop
- Students at nearby schools

We will focus our marketing efforts on these audiences because they are most likely to become regular customers at our [...]
Research Marketing Tactics
Here are some marketing tactics that we could use to reach to reach our target audiences and achieve our goals:

- Host events at our coffee shop that appeal to our target audiences (e.g., open mic nights for students or networking events [...]
- Offer discounts during slow business hours to entice [...]
- [...]

Plan Marketing Tactics
Here's how we will implement the marketing tactics above:

- Events: We will host one event per month at our coffee shop. For each event, we will send out email invitations [...]
- Discounts: Every Monday through Thursday from 3pm–6pm, we will offer $1 off any drink purchase. To promote [...]
- Local partnerships: We will reach out to five local businesses and schools to ask if we can leave stacks of coupons [...]
- Social media: [...]
- Reviews: [...]

Develop Your Timeline and Budget
This is what our marketing timeline looks like:

- Month 1: Launch email newsletter and social media accounts
- Month 2: Start distributing coupons at [...]
- [...]

Here's the budget for each marketing activity:

- Email newsletter: $50/month for MailChimp subscription
- Social media: $100/month for paid ads [...]
- [...]

Limitations and Conclusion

Fine-tuning a strong pretrained language model on 1,000 carefully curated examples can produce remarkable, competitive results on a wide range of prompts

Limitations:

- The mental effort in constructing such examples is significant and difficult to scale up.
- LIMA is not as robust as product-grade models
    - while LIMA typically generates good responses, an unlucky sample during decoding or an adversarial prompt can often lead to a weak response

The Era of 1-bit LLMs: All Large Language Models are in 1.58 Bits

Presented by: Afsara Benazir

BitNet: Scaling 1-bit Transformers for Large Language Models

Hongyu Wang*†‡ Shuming Ma*† Li Dong† Shaohan Huang†
Huaijie Wang§ Lingxiao Ma† Fan Yang† Ruiping Wang‡ Yi Wu§ Furu Wei†◇
† Microsoft Research ‡ University of Chinese Academy of Sciences § Tsinghua University
https://aka.ms/GeneralAI

**The Era of 1-bit LLMs:
All Large Language Models are in 1.58 Bits**

feb'24

Shuming Ma* Hongyu Wang* Lingxiao Ma Lei Wang Wenhui Wang
Shaohan Huang Li Dong Ruiping Wang Jilong Xue Furu Wei◇
https://aka.ms/GeneralAI

Context

Problem:

- Vanilla LLMs are in FP16
- the bulk of any LLMs is matrix multiplication - costly
- KV cache memory size

Therefore, the major computation cost comes from the floating-point addition and multiplication operations.

One solution: post training quantization – but it is suboptimal

$$H = -\sum_{i=1}^{k} p_i * \log_2(p_i)$$

To mitigate: BitNet1.58

- Represent weight values with -1, 0, and 1 requires 1.58 bits (from shannon's entropy formula, log_2(3) = 1.58
- the matrix multiplication of BitNet only involves integer addition, which saves orders of energy cost for LLMs.
- BitNet b1.58 can match full precision (i.e., FP16) baselines in terms of both perplexity and end-task performance, starting from a 3B size, when using the same configuration (e.g., model size, training tokens, etc.)

- BitNet b1.58 is based on the BitNet architecture, which is a Transformer that replaces nn.Linear with BitLinear.
- Trained from scratch, with 1.58-bit weights and 8-bit activations.

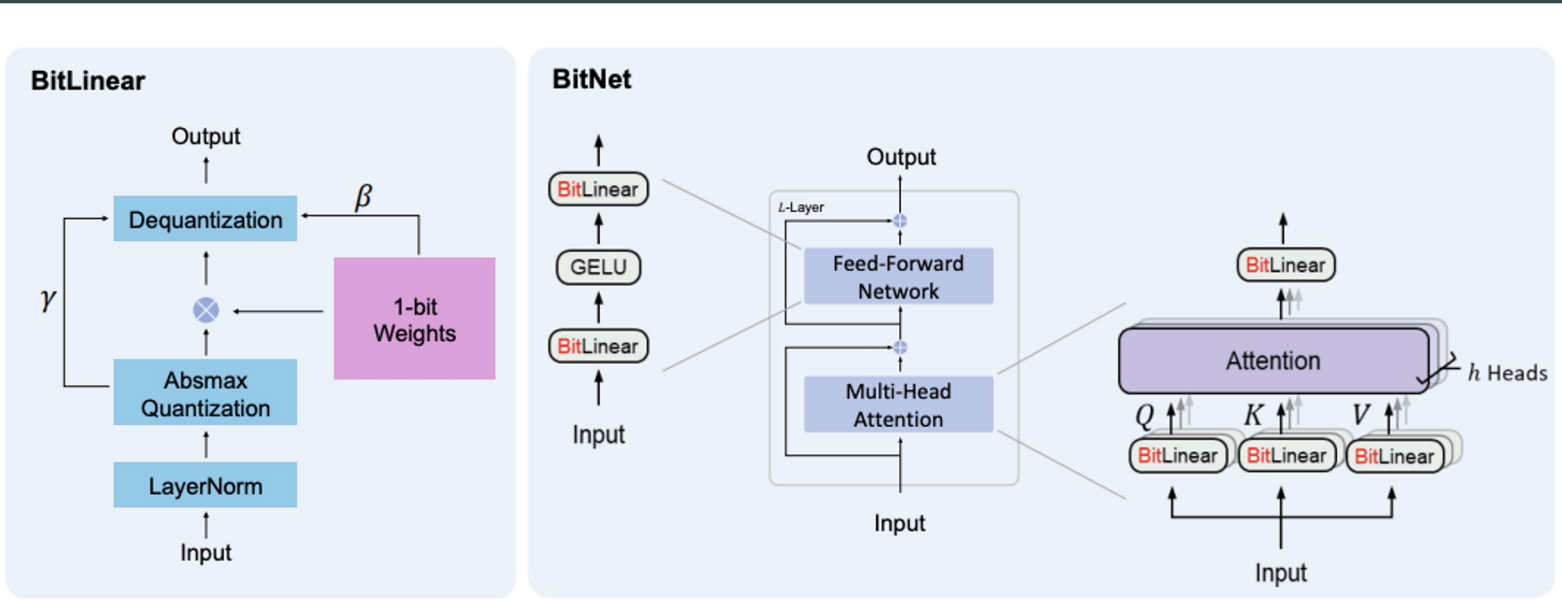
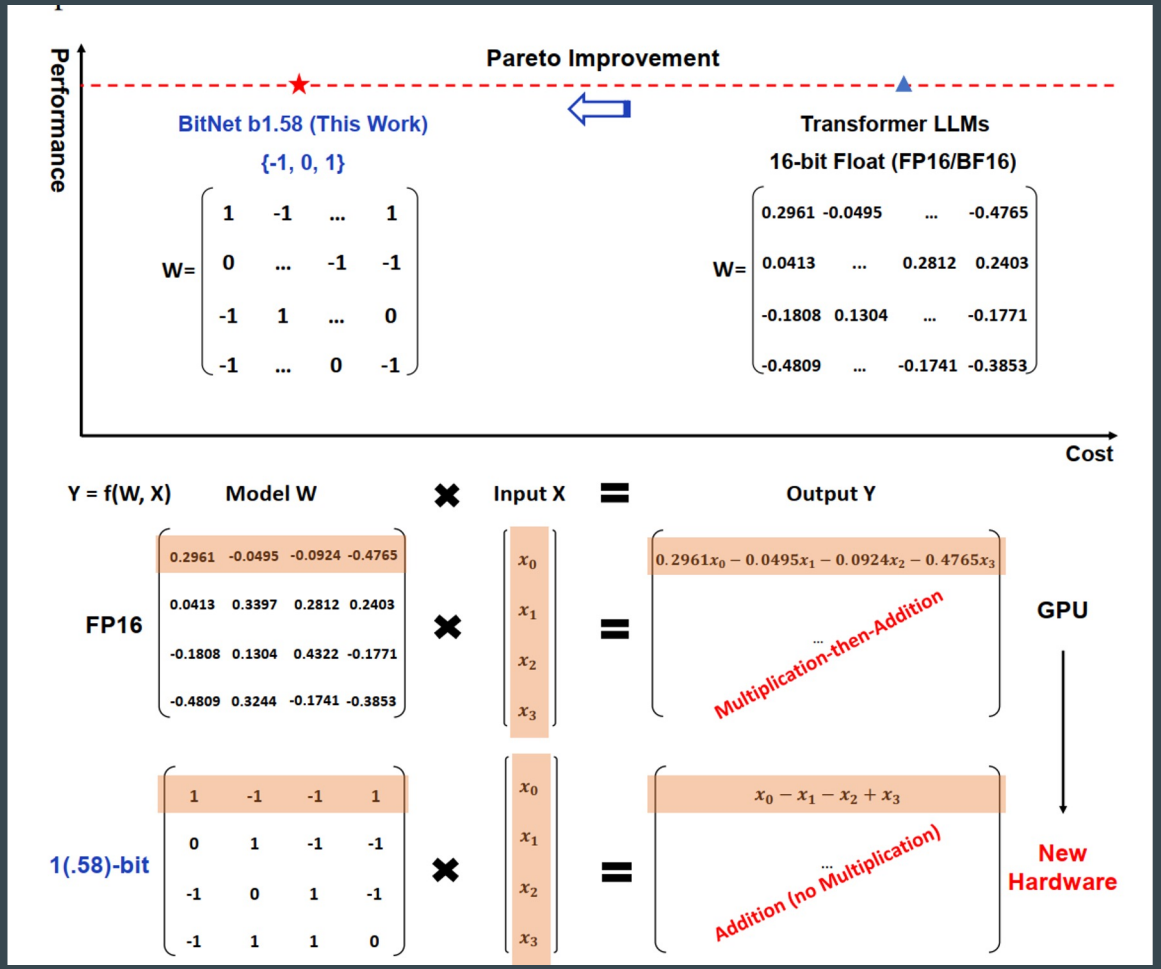
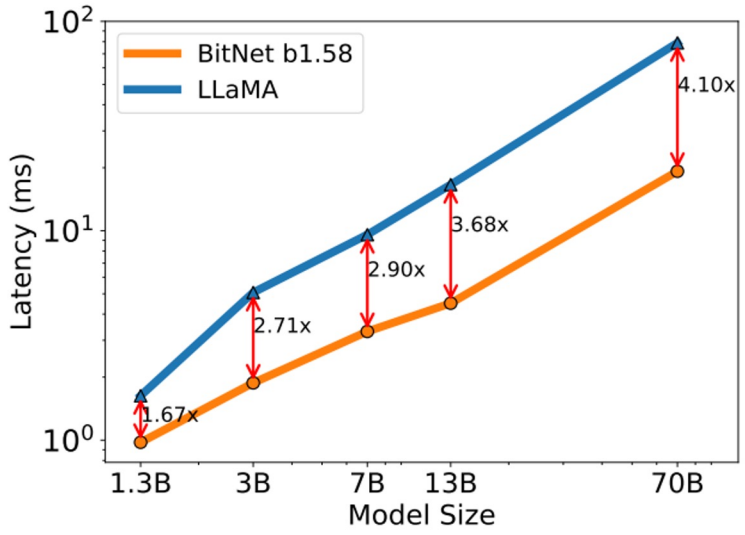


Figure 2: (a) The computation flow of `BitLinear`. (b) The architecture of BitNet, consisting of the stacks of attentions and FFNs, where matrix multiplication is implemented as `BitLinear`.


Performance
Pareto Improvement
BitNet b1.58 (This Work)
{-1, 0, 1}
W=
1 -1 ... 1
0 ... -1 -1
-1 1 ... 0
-1 ... 0 -1
Transformer LLMs
16-bit Float (FP16/BF16)
W=
0.2961 -0.0495 ... -0.4765
0.0413 ... 0.2812 0.2403
-0.1808 0.1304 ... -0.1771
-0.4809 ... -0.1741 -0.3853
Cost
Y = f(W, X)
Model W
Input X
Output Y
FP16
0.2961 -0.0495 -0.0924 -0.4765
0.0413 0.3397 0.2812 0.2403
-0.1808 0.1304 0.4322 -0.1771
-0.4809 0.3244 -0.1741 -0.3853
x_0 x_1 x_2 x_3
$0.2961x_0 - 0.0495x_1 - 0.0924x_2 - 0.4765x_3$
...
Multiplication-then-Addition
GPU
1(.58)-bit
1 -1 -1 1
0 1 -1 -1
-1 0 1 -1
-1 1 1 0
x_0 x_1 x_2 x_3
$x_0 - x_1 - x_2 + x_3$
...
Addition (no Multiplication)
New Hardware

Improvement in memory & latency



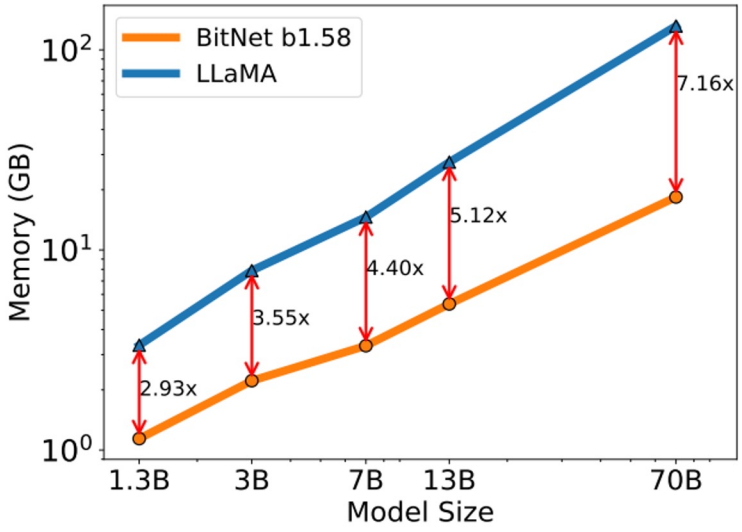
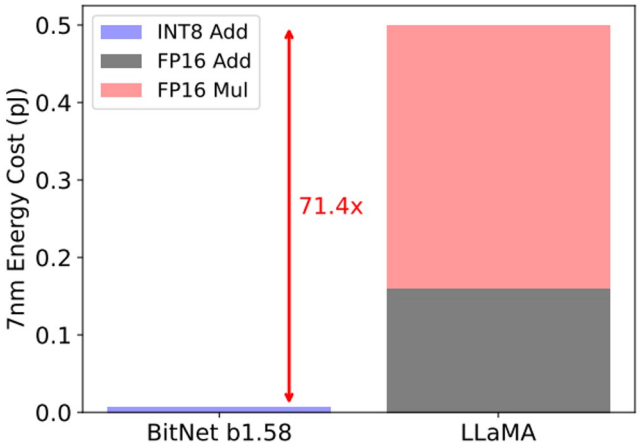


Figure 2: Decoding latency (Left) and memory consumption (Right) of BitNet b1.58 varying the model size.

Comparison: Energy consumption



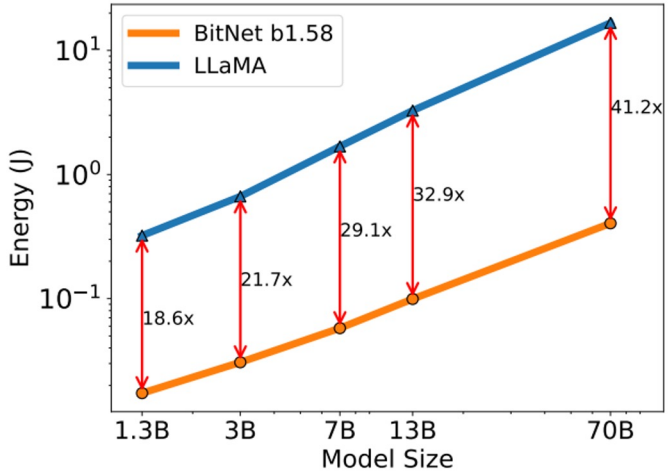


Figure 3: Energy consumption of BitNet b1.58 compared to LLaMA LLM at 7nm process nodes. On the left is the components of arithmetic operations energy. On the right is the end-to-end energy cost across different model sizes.

Performance

Models	Size	Max Batch Size	Throughput (tokens/s)
LLaMA LLM	70B	16 (1.0x)	333 (1.0x)
BitNet b1.58	70B	**176 (11.0x)**	**2977 (8.9x)**

Table 3: Comparison of the throughput between BitNet b1.58 70B and LLaMA LLM 70B.

Models	Size	Memory (GB)↓	Latency (ms)↓	PPL↓
LLaMA LLM	700M	2.08 (1.00x)	1.18 (1.00x)	12.33
BitNet b1.58	700M	0.80 (2.60x)	0.96 (1.23x)	12.87
LLaMA LLM	1.3B	3.34 (1.00x)	1.62 (1.00x)	11.25
BitNet b1.58	1.3B	1.14 (2.93x)	0.97 (1.67x)	11.29
LLaMA LLM	3B	7.89 (1.00x)	5.07 (1.00x)	10.04
BitNet b1.58	3B	**2.22 (3.55x)**	**1.87 (2.71x)**	**9.91**
BitNet b1.58	3.9B	**2.38 (3.32x)**	**2.11 (2.40x)**	**9.62**

Table 1: Perplexity as well as the cost of BitNet b1.58 and LLaMA LLM.

Future Potential

Mixture-of-Experts (MoE) Challenges and Solutions

Memory Efficiency for Long Text Processing

Innovations on Smartphones and Small Devices

Development of New Hardware for 1-bit LLMs