

LLM-e Guess: Can LLMs Capabilities Advance Without Hardware Progress?

Teddy Foley,^{*} Spencer Guo, Henry Josephson,[†] Jack Sanderson, and Anqi Qu
Existential Risk Laboratory, The University of Chicago, Chicago, IL 60637

(Dated: March 13, 2025)

This paper examines whether large language model (LLM) capabilities can continue to advance without additional compute by analyzing the development and role of algorithms used in state-of-the-art LLMs. Motivated by regulatory efforts that have largely focused on restricting access to high-performance hardware, we ask: Can LLMs progress in a compute-constrained environment, and how do algorithmic innovations perform under such conditions?

To address these questions, we introduce a novel classification framework that distinguishes between compute-dependent innovations which yield disproportionate benefits at high compute levels (e.g., the Transformer architecture and mixture-of-experts models) and compute-independent innovations, which improve efficiency across all compute scales (e.g., rotary positional encoding, FlashAttention, and layer normalization). We quantify these contributions using the Compute-Equivalent Gain (CEG) metric, which estimates the additional compute that would be required to achieve similar improvements without these algorithmic advancements.

To validate this framework, we conduct small-scale training experiments with a scaled-down GPT-2 model. Our results confirm that compute-independent advancements yield meaningful performance gains even in resource-constrained settings, with efficiency improvements of up to 3.5 over a baseline model. By contrast, compute-dependent methods, such as sparse attention, provide little benefit or even degrade performance at small scales, reinforcing the importance of compute availability for certain algorithmic gains.

These findings indicate that while restrictions on hardware may slow certain LLM gains, they cannot prevent all gains driven by algorithmic advancements. From a policy perspective, this challenges the assumption that limiting access to compute is a sufficient control mechanism for AI capabilities. Instead, effective governance should also consider monitoring and shaping algorithmic research. Additionally, our framework offers a practical tool for researchers to predict the scalability of algorithmic improvements and optimize experimental design. By clarifying the distinct roles of hardware scaling and algorithmic progress, this study provides insights for AI forecasting, investment strategies, and regulatory approaches in a landscape where progress is increasingly driven by both hardware and algorithms.

^{*} Author order is alphabetical; all authors contributed equally.

[†] Correspondence to henryj@uchicago.edu

CONTENTS

I. Introduction & Motivation	3
II. Dissecting Algorithmic Progress: Compute's Role in Algorithmic Evolution	4
A. DeepSeek V3: A Case Study	4
B. Quantifying Compute-Equivalent Gain	4
C. Compute-Dependent Algorithms	6
D. Compute-Independent Algorithms	9
E. Results	11
III. Empirical Insights: Evaluating Algorithmic Gains	11
A. Experimental Details	11
B. Results	12
IV. Discussion	13
A. Implications for Policymakers	14
V. Conclusion	15
References	15

I. INTRODUCTION & MOTIVATION

The rapid advancement of Large Language Models (LLMs) has been driven by two key factors: increasing computational resources and algorithmic improvements. However, the relative contributions of these two drivers remain an open question with significant implications for AI progress, regulation, and forecasting. If compute scaling is the dominant force behind LLM improvements, then restricting access to advanced hardware - through export controls or regulatory measures - could meaningfully slow AI development. Conversely, if algorithmic innovations can drive substantial progress even in a compute-limited environment, then such restrictions may be far less effective than anticipated.

AI regulation has largely focused on hardware controls, particularly through export restrictions on cutting-edge chips.¹ Given that many recent LLM advancements have stemmed from pure scaling,² these restrictions prompt two fundamental questions. First, if computational power were frozen at current levels, could we still expect LLMs to continue improving? This speaks to the potential for progress even in a world where access to greater compute is fully restricted. Second, how do algorithmic or architectural advancements translate from low compute to high compute, and would frozen compute affect this translation? This question is more practical; rather than assuming perfect enforcement, it considers whether export controls could successfully keep certain innovations from being leveraged at scale.

Existing research³⁻⁶ has attempted to estimate the role of algorithmic progress through scaling laws, which describe how performance improves with more compute. However, these approaches have key limitations: scaling laws do not directly measure the contributions of specific algorithmic innovations and may understate the role of software-driven advancements that enhance efficiency at all compute levels. To address this, we introduce a new analytical framework that distinguishes between compute-dependent and compute-independent algorithmic advances. Compute-dependent advancements yield benefits primarily at large compute scales, whereas compute-independent advancements enhance efficiency across all scales. We pair this classification with Compute-Equivalent Gain (CEG) estimates to quantify the extent to which algorithmic improvements depend on compute. Through a case-study-driven approach, we identify and analyze major algorithmic innovations from the past decade, validate our framework with small-scale experiments, and explore the broader implications for AI research and governance.

Our findings suggest that while many of the most impactful innovations - such as the Transformer architecture and Mixture-of-Experts models - have been compute-dependent, there also exist critical compute-independent advancements, such as Rotary Positional Encoding (RoPE), FlashAttention, and Layer Normalization. These innovations improve model efficiency even in resource-constrained settings, indicating that AI progress can continue even in a compute-limited environment. Our experimental results further confirm that compute-independent techniques provide meaningful performance gains in small-scale models, while compute-dependent methods offer little benefit or even degrade performance at lower compute levels.

Beyond geopolitical concerns, these questions are also central to AI regulation as a whole, particularly in ensuring safe and aligned AI development. If major capability gains can arise from algorithmic improvements alone, then effective governance may require more than just controlling hardware - it may also necessitate monitoring and shaping the trajectory of algorithmic research, a far more challenging task. Many current regulatory frameworks assume compute is the primary bottleneck⁷ and focus on restricting access to large-scale infrastructure. However, if algorithmic innovations continue to drive progress independently of hardware, then safety strategies and governance approaches will need to evolve accordingly.

This also has implications for AI forecasting and investment. Many companies and institutions have placed large bets on hardware, assuming that scaling compute will remain the primary driver of future breakthroughs. However, if algorithmic efficiency gains remain significant, AI advancements may occur sooner than expected, even under compute constraints. Understanding whether AI progress is fundamentally hardware-driven or whether algorithmic ingenuity alone can sustain rapid improvements will be crucial for predicting AI's trajectory.

By addressing these questions, this paper provides a clearer framework for policymakers, researchers, and

industry leaders. Our findings could inform more robust AI governance strategies - ones that remain relevant not just in today's landscape of export controls and compute thresholds, but in a future where AI progress may be increasingly driven by factors beyond just compute.

II. DISSECTING ALGORITHMIC PROGRESS: COMPUTE'S ROLE IN ALGORITHMIC EVOLUTION

To explore the effectiveness of hardware controls, we ask: what if AI development had to progress without more hardware? Suppose datacenter construction halted, no new GPUs or TPUs were manufactured, and researchers had to work with today's infrastructure—how much further could AI improve?

At first glance, it is tempting to assume that progress would slow dramatically, but there is also reason to believe algorithmic advancements could drive significant gains independent of compute increases. Even with fixed hardware, there are still untapped opportunities. Better prompts, more effective evaluation strategies, and optimized architectures could let us achieve more with the same resources.

To quantify this, we look at past algorithmic advancements: how much recent AI progress is due to better algorithms rather than mere compute scaling? To this end we use the "compute-equivalent gain" framework, measuring AI advancements that came from algorithmic improvements alone. Individual case studies, such as DeepSeek, provide concrete data. If we believe DeepSeek's numbers, algorithmic advancements alone can yield substantial improvements, even in a world with frozen compute. This has real-world policy implications. Export controls, for example, are designed to slow AI progress by restricting hardware, but if software improvements continue to drive significant capability gains, how much of a brake can hardware restriction really impose? In this section we explore these questions, aiming to untangle how far AI can go from algorithmic progress alone and whether hardware can truly set limits on LLM advancements.

A. DeepSeek V3: A Case Study

As a preliminary case study, we look at DeepSeek V3, a model that notably required only 2.788M H800 GPU hours for its full training.⁸ In comparison, LLaMa 3.1-405B-Instruct achieved comparable or worse performance with 30.84M GPU hours⁹ on superior H100 GPUs.¹⁰ This clearly suggests that, if compute advances were to suddenly stop, algorithmic improvements in model training could still increase capabilities. Perhaps more pertinently, DeepSeek V3 also demonstrates that export controls cannot completely prevent the targeted countries from developing their own near-frontier AI models.¹¹ Of course, some have cast doubt on whether DeepSeek's training numbers are reliable. Ultimately, however, even DeepSeek CEO stated that "Money has never been the problem for us; bans on shipments of advanced chips are the problem".¹² Either way, the algorithmic advances used in DeepSeek V3 such as multi-headed latent attention, mixture-of-experts, and mixed-precision training⁸ no doubt contributed to its superior performance. DeepSeek's performance, then, suggests that even the most effective hardware controls cannot completely halt the improvement of LLMs in areas export controls affect.

B. Quantifying Compute-Equivalent Gain

The development of LLMs has been shaped by both hardware innovations and algorithmic breakthroughs, evolving in tandem to enable unprecedented scaling and efficiency. Early hardware innovations — notably, NVIDIA's introduction of CUDA in 2006 — made it feasible to do general-purpose computing on GPUs, which laid the foundation for deep learning applications. Throughout the 2010s, specialized AI hardware,

such as Google’s Tensor Processing Units (TPUs) (2013-2016), and distributed training frameworks like Microsofts Project Adam (2015) further expanded computational capabilities. By 2018, the emergence of large-scale distributed deep learning, supported by hardware like NVIDIA’s Tesla V100 and TPU v3 pods, enabled the training of increasingly complex models.

In this section, we analyze some of the primary algorithmic advancements that have driven LLM progress. For each algorithmic advancement, we review the algorithm and its applications, classify the advance into “compute-independent” or “compute-dependent”, and finally estimate the algorithm’s compute-equivalent gain.

1. Compute-Independent vs. Compute-Dependent Algorithmic Advances

We now introduce the framework of compute-independent vs. compute-dependent algorithmic advances, following our schema in Table I. We compare the performance of a new algorithm (e.g., training procedure or architectural innovation) to its predecessor (i.e., status quo) algorithm at low and high levels of compute. If an algorithm yields similar performance gains over a previous algorithm at both low and high use of compute, we interpret it as a compute-independent advance. Alternatively, if an algorithm provides small benefits (or worsens performance) at low compute but gives large benefits at high compute, we classify it as a compute-dependent advance.

TABLE I. Schematic for understanding compute-independent (I) and compute-dependent (D) algorithmic advances.

	Status Quo Algorithm	New Algorithm (I)	New Algorithm (D)
Low Compute	Slow	Faster	Slow
High Compute	Slow	Faster	Very Fast

We note that these two categories naturally separate the types of compute-equivalent gains into discoveries that occur under low-resource or high-resource conditions, but they do not include algorithms whose consequence (rather than prerequisite) is high compute. We thus note the existence of a third “compute-unlocking” category of algorithms which typically both depend on *and* enable scaling, such as parallelization frameworks. These types of algorithms do not generally provide a compute-equivalent gain—their primary benefit is to use compute more effectively and enable scaling. As such, they are less relevant to our thesis, and we accordingly do not discuss them here. Naturally, there are limitations to our approach, which we outline in Section IV.

2. Compute-Equivalent Gain

The concept of compute-equivalent gain (CEG) is introduced in Davidson *et al.*^[13] as “how much additional training compute would have been needed to improve benchmark performance by as much as the post-training enhancement.” In order to calculate CEG, we roughly estimate compute cost using the following proportion:

$$\text{Compute Cost} \propto \text{Active Parameters Per Step} \times \text{Training Steps} \quad (1)$$

Given the compute cost of a baseline model (C_b) and of a equally performant but more efficient model (C_e), we calculate the CEG as:

$$\text{Compute Equivalent Gain} = \frac{C_b}{C_e} \quad (2)$$

C. Compute-Dependent Algorithms

1. Transformer

The single most influential algorithmic advancement in the last 10 years is undoubtedly the transformer architecture.¹⁴ In fact, some estimates⁵ have suggested that the transformer architecture by itself accounts for nearly 20% of language modeling improvements since 2015.

The primary breakthrough offered by the transformer is the self-attention mechanism. Self-attention allows the transformer model to draw global dependencies between input and output by tracking how each input token affects each of the others¹⁵ in a constant number of sequential operations.¹⁶ Previous state-of-the-art models, such as recurrent neural networks, would require $O(n)$ sequential operations on top of any other scaling; the introduction of the transformer architecture thus allowed much greater parallelization in training.

While this parallelization ability is a great advantage, it ultimately requires a large amount of memory to store all the states of the model. Transformers often include multiple encoder or decoder blocks (or both), with each block containing at least a self-attention layer and feed-forward layer. While it is possible that the transformer architecture could have been invented before sufficient compute was available to train models, its major benefits emerge only once sufficient compute exists to hold the larger models. Previous research has also found that other architectures, like Long-Short Term Memory models, are more efficient than transformers at smaller scales, but the transformer improves as the number of parameters and amount of data increases.¹⁷ These facts together show that the transformer is compute-dependent: its advantages emerge only in much larger models.

We estimate the CEG of the transformer directly from Vaswani *et al.*^[14] which showed that the transformer performed as well as or better than all other state-of-the-art models on English-to-French and English-to-German translation tasks. The most performant English-to-French non-transformer model required 7.7×10^{19} FLOPs to train, compared to the base transformer’s 3.3×10^{18} , indicating that the transformer offers at least a $10\times$ CEG. Further, the big transformer with 2.3×10^{19} FLOPs achieved similar performance to the ConvS2S Ensemble at 1.2×10^{21} FLOPs. This would put the transformer’s CEG upwards of $50\times$, which is not unreasonable given previous findings about the transformer’s influence relative to other algorithmic improvements.⁵

2. Sparse Attention

As previously discussed, one of the main drawbacks of the traditional self-attention mechanism is its runtime. Because each token interacts directly with every other token, the runtime of the mechanism scales quadratically with sequence length ($O(n^2)$). While this issue is not of major concern with shorter sequences, it becomes a very pertinent issue once processing long sequences becomes the focus. To combat this problem, sparse attention¹⁸ is a strategy that limits the size of the attention window, thus reducing runtime and complexity.

The idea of factorized self-attention is as follows: Given a set $S = S_1, S_2, \dots, S_n$ where each $S_i \in S$ is the set of indices of inputs that attend to the i th input, traditional self-attention (specifically causal/masked self-attention) defines $S_i = \{j : j \leq i\}$; every previous token affects the next token. Factorized self-attention, rather, is only interested in a subset of the indices $S_i \subset \{j : j \leq i\}$ where $|S_i|$ scales with n . To address the possibility of disconnectivity within this attention mechanism, each of the p attention heads is set such that for every $j \leq i$ token pair, i can attend to j in $p + 1$ steps. That is, given a sequence (j, a, b, c, \dots, i) , head 1 will ensure j attends to a , head 2 will ensure a attends to b , and so on. This guarantees that at least indirectly, every token can still affect future tokens. This combined with the indices that tend to each future token scaling with \sqrt{n} means that sparse attention can scale in a much better $O(n\sqrt{n})$ time.

Sparse attention demonstrated equivalent or improved performance over standard transformers on various long-sequence modeling tasks, ranging from image generation to NLP tasks (see Section 7 in¹⁸). While the compute scale used at the time was lower than what might be used today, we argue that sparse attention’s reduced capacity would lead to a degradation in model performance at low compute levels. At small scales, leaning towards less sparsity means especially negligible benefits, whereas leaning toward more sparsity means there may simply not be enough information captured. Our experiments (Section III B) appear to validate these claims. We thus classify sparse attention as a compute-dependent advancement.

When trained on the EnWiki8 dataset, the sparse transformer achieved 0.99 bits per dimension, matching that of the state-of-the-art model trained with more than double the number of parameters¹⁸ at half the time-per-iteration (1.33 vs. 0.55). With half the active parameters per step and roughly half the training steps, the sparse transformer’s CEG is thus $2 \cdot \frac{1.33}{0.55} = 4.84\times$ of contemporaneous state-of-the-art models.

3. Mixture of Experts

The mixture of experts (MoE) architecture subsumes several variants and algorithmic developments that have been introduced over the last decade, most prominently in the form of the MoE LSTM¹⁹ and Switch Transformer,²⁰ which we briefly review here. We also discuss the DeepSeekMoE architecture²¹ which is employed in DeepSeek-v3 (see II A). In essence, the MoE architecture increases the effective number of parameters in an architecture without significantly increasing compute by using a group of individual “expert” models which are combined via a “routing” network. Each expert is activated during a forward pass of the model, depending on the input token and the routing network’s choice. Because only one (or a few) experts are activated on a given token—even though the architecture typically contains many experts, on the order of dozens to hundreds—the total number of parameters (and hence capabilities) of the model can be very large without increasing the runtime on a given token. Although the general idea is relatively old and straightforward to explain, dating back to the 90s²², it requires many innovations in loss functions and computing infrastructure (due to the increased communication requirements, for example) to make it fully operational. These algorithmic innovations, detailed below, have been critical to the MoEs success in modern LLMs such as DeepSeek-v3.

The LSTM-based MoE model in Shazeer *et al.*^[19] employed the expert and routing networks immediately after an LSTM layer, and this scheme was later transferred to transformer architectures where the feedforward network after each self-attention layer is replaced by a MoE layer (see e.g. Figure 2 in Fedus *et al.*^[20]). The choice of expert networks is determined by the weights of the routing network using a softmax distribution over the N experts, from which only $K \ll N$ are chosen. The number of experts activated may be small or even just one,²⁰ reducing the computation at each step to effectively the cost of a single expert network. The routing network is also encouraged to balance the tokens across the experts using an auxiliary loss function during training, although this was later suggested to be unnecessary.^{21,23} More recently, in the DeepSeekMoE architecture, the authors identify that segmenting the experts further makes each expert more specialized, reducing the overlapping knowledge each one contains and increasing the flexibility of the model.²¹ They further designate a fixed set of expert networks as “common” (i.e., always activated during a forward pass), with the goal of mitigating redundancy between the experts.

Overall, the main advantage of the MoE architecture is an increase in the effective parameter count of the model without a concomitant increase in compute (FLOPs). In principle, this means that one can scale to larger models with equivalent (or lower) compute requirements, while improving parameter efficiency through specialization. For example, in Fedus *et al.*^[20] the authors perform a series of experiments holding FLOPs per token constant with a smaller transformer model (T5) having 200M parameters vs. the equivalent Switch Transformer having 7B parameters (a $35\times$ difference), showing the Switch Transformer significantly outperforms the base transformer model on language modeling tasks. We note that even the FLOPs/token in the smallest experiments are approximately 3 orders of magnitude higher than the experiments in the original LSTM version (Figure 2 in Shazeer *et al.*^[19]), which observes similar benefits for the MoE layer. In the Switch Transformer work, the authors conduct scaling experiments to test the consistency of the

gains across a wide range of compute configurations, noting that the MoE architecture is highly amenable to parallelization across a number of GPUs by splitting the expert parameters across hardware units.

These findings are generally confirmed by DeepSeek’s experiments with their DeepSeekMoE architecture,²¹ where they successfully train a MoE model at a 2B parameter scale up to 671B parameters (37B activated) with the DeepSeek-V3 model.⁸ At the small scale, they find that the 2B parameter model achieves comparable performance with GShard 2.9B,²⁴ which has $1.5\times$ the expert parameters and computation, while the 16B-parameter model achieves comparable performance with LLaMA2 7B²⁵ while requiring only 40% of the compute. Together, these findings imply that the MoE algorithms are helpful at multiple compute scales, but that the majority of the algorithmic gains are found in large models where one can take advantage of parallelization. At the same time, there is evidence that MoE algorithms are not extremely compute-dependent, given that early practical versions were discovered in Shazeer *et al.*^[19] before the massive scaling increases enabled by transformers. Overall, we classify these developments as compute-dependent, as MoE-based models display their largest compute-equivalent gains in the scales over 10^{23} FLOPs, and as a result are primarily deployed as alternatives to the largest dense transformer models (e.g. DeepSeek-v3 vs. Claude 3.5 or GPT-4o).

Once again, based on the 2.788M H800 hours required to train DeepSeek-V3⁸ compared to the 30.84M H100 hours to train LLaMa 3.1-405B-Instruct’s,^{9,10} given their similar performance we can estimate the CEG offered by DeepSeek-V3 as roughly $11\times$. There are a number of confounding factors, such as DeepSeek’s slightly superior benchmark metrics and worse GPUs, but also the model’s supplementary algorithmic advances, e.g. mixed-precision training. We see in the Switch Transformers paper that a FLOP-matched Switch Transformer offers a roughly $7\times$ training speedup over a base model²⁰ and thus an equivalent CEG. This falls reasonably in line with DeepSeek-V3’s CEG and is likely a realistic estimate for the compute-equivalent gain offered by a MoE architecture. It should be noted, however, that the MoE architecture does not fit cleanly into our CEG framework. Since the total number of parameters is large, but only a subset is active per step, the compute required for a single forward pass is relatively low, while the compute needed to store and manage the entire model remains high. This distinction serves as another reason for our classification of this advancement as compute-dependent

4. Multi-Query Attention

To reduce memory usage in the attention mechanism, multi-query attention (MQA) modifies standard multi-head attention by having all query heads share the same keys and values.²⁶ This reduces the KV cache size by a factor of h (the number of attention heads) while maintaining multiple query heads, which roughly achieves comparable model quality to standard multi-head attention (albeit with minor degradation). For example, on the WMT14 English-to-German translation task, MQA achieved a BLEU score of 28.5 with beam-4 decoding, nearly matching the baselines 28.4, with only a slight increase in perplexity (1.439 vs. 1.424).²⁶

At the time of its introduction in 2019, MQA was tested primarily on small models, often non-transformers, where memory constraints were not a major concern. As a result, its benefits were not immediately apparent. However, as model sizes grew, memory efficiency became increasingly important, making MQA a crucial optimization in modern LLMs (including Falcon, PaLM, and StarCoder). We thus classify it as a compute-dependent advancement. Its widespread adoption is largely driven by the demands of scaling, which further reinforces its classification as a compute-dependent technique.

Interestingly, while the training time with and without MQA remains comparable ($13.0\mu s$ vs $13.2\mu s$ per token), it dramatically reduces the memory bandwidth requirements during inference. The encoder inference time was reduced from $1.7\mu s$ to $1.5\mu s$ per token, while the decoder inference time was reduced from $46\mu s$ to $3.8\mu s$ per token.²⁶ This gives us a modest CEG estimate of $1.13\times$ for encoder tasks, but $12.1\times$ for decoder tasks. Implementing multi-query attention is thus very valuable for inference-heavy workloads, but less important for training itself.

D. Compute-Independent Algorithms

1. Rotary Positional Embedding

The self-attention mechanism models interactions between tokens in a sequence without regard to their order - a property known as permutational equivariance. As a result, the algorithm does not inherently “know” whether a token appears at the beginning or end of a sentence. However, since natural language is inherently sequential, this issue is typically addressed using positional embeddings, which encode information about a tokens location within a sequence.

In the original Transformer paper (see Transformer II C 1), positional encoding was implemented using periodic functions (sines and cosines) with geometrically spaced frequencies up to a fixed maximum sequence length (10,000).¹⁴ While simple and efficient, this approach has two major limitations that led to the development of Rotary Positional Embedding (RoPE). First, absolute encoding represents a tokens position in isolation rather than encoding its relative distance to other tokens, making it less effective for capturing relationships in long sequences. Second, the use of a fixed maximum sequence length imposes constraints on how much text the model can process effectively, leading to degradation in quality when modeling long-range dependencies.

To circumvent these issues, RoPE²⁷ was introduced in 2023 as a way to model relative sequence positions through rotation matrices rather than additive embeddings. As the name suggests, the positional encoding is generated by rotating the initial token embedding vectors depending on their sequence position and then computing the relative distance by multiplying the two vectors together. Because of the mathematical properties of rotation matrices, the inter-token dependency naturally decays as the sequence distances increase, matching the modeling objectives. Moreover, RoPE can easily be implemented with sparse matrices, requiring no increases in compute compared to the absolute encoding scheme from Vaswani *et al.*^[14].

RoPE presents a clear example of a compute-independent algorithmic advance, as the improvements can be observed in all scales of transformer architectures. RoPE has been incorporated in many language models after its original publication, and has been further extended to handle even longer sequence lengths (e.g. see LongRoPE²⁸ or AliBi²⁹). One can view it as essentially a step improvement in context length, i.e. it achieves the same performance for a much larger number of tokens and provides a greater ability to summarize long documents.

We estimate RoPE’s compute-equivalent gain directly from benchmarks in the original paper. The RoFormer paper trained BERT (110M base parameters³⁰) and RoFormer (65M base parameters) over 100K training steps. Both models exhibited comparable performance on downstream GLEU tasks. Consequently, we estimate the computational cost for each model as $C_{\text{BERT}} = (110 \times 10^6) \cdot (1 \times 10^5) = 1.1 \times 10^{13}$ and $C_{\text{RoFormer}} = (65 \times 10^6) \cdot (1 \times 10^5) = 6.5 \times 10^{12}$. Thus, RoPE’s estimated CEG is $\frac{1.1 \times 10^{13}}{6.5 \times 10^{12}} \approx 1.7 \times$.

2. FlashAttention

Modern LLMs are build using the transformer architecture, and as previously discussed, the memory consumption of a naive implementation of self-attention on n tokens scales in $O(n^2)$. Such quadratic scaling can easily become prohibitive when handling long sequences required for complex language processing tasks.

FlashAttention³¹ is an algorithm developed to take advantage of the physical memory layout on the GPUs used for training. In a modern GPU, the computing speeds are significantly faster than the speeds at which data is transferred between the memory storages, meaning that only a fraction of time is actively spent doing the attention calculation during a typical SA calculation. In the FlashAttention paper, the authors leverage

this difference in speed to compute the attention matrix blockwise (i.e. in chunks, rather than all at once) rather than transferring to and from the slow (but large) GPU memory. The result of these hardware-aware optimizations is approximately a $2\text{-}4\times$ speedup compared to the default PyTorch implementation of attention and enables sequence lengths around 4 times the original. Later iterations of FlashAttention further develop hardware-based optimizations to increase the speed and long-sequence capabilities^{32,33}. FlashAttention is now the primary algorithm used in state-of-the-art language modeling due to its performance efficiency and their widespread use of attention mechanisms.

FlashAttention shows time and memory improvements at all sequence lengths tested (see Fig. 3 in Dao^[32]). This is also generally true for later generations, suggesting that the algorithmic innovation was largely compute-independent. In this specific case, it appears that algorithmic optimization can significantly reduce compute requirements without sacrificing model accuracy or capabilities. However, we identify two confounding factors in our analysis. First, FlashAttention takes special advantage of the hardware layout found on modern GPUs (specifically the multiple levels of memory between the fast SRAM and slow high bandwidth memory), meaning that it is highly dependent on architectural constraints and, more generally, the ubiquity of GPUs for AI training. The second factor, which arises primarily in comparing the performance of FlashAttention 2 and FlashAttention 3, is slight differences in the implementation of the algorithms, most notably the use of low-precision float point formats such as bfloat16 or FP8.³⁴ As discussed in the Mixed Precision section, these lower-precision formats have become much more common for accelerating training and inference which makes a direct comparison between different FlashAttention variants challenging. In the end, the widespread use of FlashAttention in almost all transformer architectures points to the algorithm being primarily untethered from compute increases.

As discussed above, FlashAttention’s hardware optimizations offer roughly a $2\text{-}4\times$ speedup compared to PyTorch’s attention. We take this as our estimate of the CEG offered by FlashAttention.

3. Layer Normalization

Prior to layer normalization, batch normalization was the standard for training neural networks.³⁵ Batch normalization consists of computing the mean and variance of each feature within a mini-batch, then using these statistics to normalize the features for each input. For NLP uses, however, batch normalization has a significant disadvantage in that the mini-batch size is not always fixed, meaning that the computed statistics may not be representative. Despite this drawback, batch normalization provides a significant reduction in the training time of neural networks that utilize the technique.

Layer normalization provides similar training speedups in a more reliable manner for NLP tasks. Rather than using the mean and variance of each minibatch to normalize, it uses the mean and variance of the features within a single input.³⁵ Layer normalization is therefore not prone to being influenced by small mini-batch sizes. Additionally, as demonstrated in the paper, layer normalization empirically provides a great training speedup to neural networks, especially when given long sequences or small mini-batches.

Given that layer normalization demonstrated speedups on the much smaller neural networks used in 2016 and is still often used in models today (or is swapped out for similar normalization techniques, such as RMSNorm), this algorithmic advancement falls into the compute-independent category. This is further supported by the fact that the speed-up provided by layer normalization was mathematically proven in Ba *et al.*^[35]; no advances in compute were needed to demonstrate the improvement this algorithm provides.

We estimate the CEG offered by layer normalization directly from the layer normalization paper, which demonstrates the convergence of an LSTM with layer norm occurring 60% faster than without it³⁵. Thus, layer normalization’s CEG is approximately $1.67\times$.

E. Results

We summarize our findings in Table II. We see clearly that numerous algorithmic advancements have provided non-trivial increases in compute-equivalent gain, implying that if compute power were to be frozen tomorrow, LLM progress would still continue. Furthermore, this is looking purely at algorithmic progress, ignoring, e.g., the new paradigm in using compute at inference time via chain-of-thought, among other techniques. We additionally observe that compute-dependent advances tend to provide greater compute-equivalent gain than compute-independent advances, suggesting that compute plays a non-trivial role in furthering algorithmic progress.

TABLE II. Summary of estimated CEG for reviewed algorithmic advancements.

Advancement	CEG	Dependent/Independent
Transformer	10–50×	Dependent
Sparse Attention	4.84×	Dependent
MoE	7–11×	Dependent
MQA	1× for training 1.13× for encoder tasks 12.1× for decoder tasks	Dependent
RoPE	1.7×	Independent
FlashAttention	2–4×	Independent
LayerNorm	1.67×	Independent

III. EMPIRICAL INSIGHTS: EVALUATING ALGORITHMIC GAINS

When an algorithmic advance improves performance at low compute, whether by enabling smaller models to perform better or reducing the need for extensive training data, how should we expect those gains to translate at large compute scales? Some improvements may scale linearly as model size increases, while others might exhibit nonlinear effects, compounding in unexpected ways. This scaling behavior determines the true “compute-equivalent gain” of an algorithmic breakthrough.

A. Experimental Details

In order to empirically verify our predictions in Section II, we conducted a series of experiments using small models to measure the effects of different algorithmic improvements. The goal is to understand how these effects extend when scaled to larger models: whether the gains persist, amplify, or plateau. Due to monetary constraints, we were unable to run large-scale experiments, however we find that the smaller-scale was still more than enough to validate our framework.

For these experiments, we chose to train a smaller-scale version of OpenAI’s GPT-2 model based on the PyTorch implementation nanoGPT. The hyperparameters for the architecture and training are listed in Table III. Due to limited computing resources and time constraints, we trained our model on the OpenWebText dataset for a fixed number of iterations (50,000) rather than to convergence, which requires approximately 600,000 iterations for the full-size nanoGPT model. We implemented each algorithm using the native PyTorch libraries (LayerNorm, FlashAttention) or as closely to the original report as possible otherwise (MQA, RoPE, and Sparse Attention). Given the small size of our model, we were unable to implement MOE, so that particular algorithm is left out of our experimental results.

For each experiment, we applied one of the algorithms and measured the model’s cross-entropy validation loss after the set number of training iterations. We also record the GPU utilization (or MFU, model FLOP

utilization) to compute the total number of FLOPs used during training. We estimate two metrics: the overall performance gain (as measured by the validation loss) after completing training (i.e., after 50,000 iterations), and the compute equivalent gain compared to the baseline at 25,000 and 50,000 iterations. Concretely, we identify the iteration at which a given algorithm achieves a validation loss equal or lower than the baseline loss at 25,000 or 50,000 iterations. We then estimate the (relative) total FLOP usage by multiplying the number of training iterations required by the average MFU (assuming that the MFU is approximately constant during training, which is generally true). Our code to reproduce experiments is available at <https://github.com/tedfoley/nanoGPT>.

TABLE III. Training hyperparameters for experiments compared with GPT-2

Hyperparameter	Original (GPT-2)	Ours
<i>Model Architecture</i>		
Number of layers	12	8
Number of heads	12	8
Embedding dimension	768	512
Block size	1024	512
Dropout	0.0	0.2
<i>Batch Size Configuration</i>		
Batch size	12	64
Gradient accumulation steps	40	8
Total effective batch size	~491k tokens	~262k tokens
<i>Training Schedule</i>		
Maximum iterations	600k	50k
Warmup iterations	2000	1000
Learning rate decay iterations	600k	50k
<i>Optimizer</i>		
Learning rate	6×10^{-4}	8×10^{-4}
Minimum learning rate	6×10^{-5}	8×10^{-5}
β_1	0.9	0.9
β_2	0.95	0.95
Gradient clipping value	1.0	1.0
<i>Regularization</i>		
Weight decay	0.1	0.1

B. Results

We summarize our findings in Table IV, with training and validation loss curves shown in Figure 1. At 50,000 iterations, both the training and validation losses are decreasing, indicating we have not yet saturated our model’s capacity. We find that compared to the baseline, the only algorithms which show significant CEG at the low-compute scale of our experiment were those that we had categorized as compute-independent: LayerNorm (2.24–2.35) RoPE (1.42–1.54), and FlashAttention (0.91). To achieve the highest possible CEG, we ran a final training run with a combination of these algorithms, which resulted in a CEG of 3.03–3.14. Naively multiplying the CEG of each algorithm independently gives us an estimated CEG of 2.37–2.79, demonstrating that at least for these three specific algorithms, each CEG is mostly independent from the others, but that there may be small cooperative effects between them.

Of the two compute-dependent algorithms we tested, MQA (CEG of 0.91) performed comparably to the baseline,

[INSERT NEW SPARSE ATTENTION RESULTS]

While sparse attention is compute-dependent, its benefits are largely dependent on the size of the context window and . Sparse attention solves the problem of traditional self-attention’s quadratically scaling runtime as the sequence length increases. However, our small model had a short context window of just 512 tokens

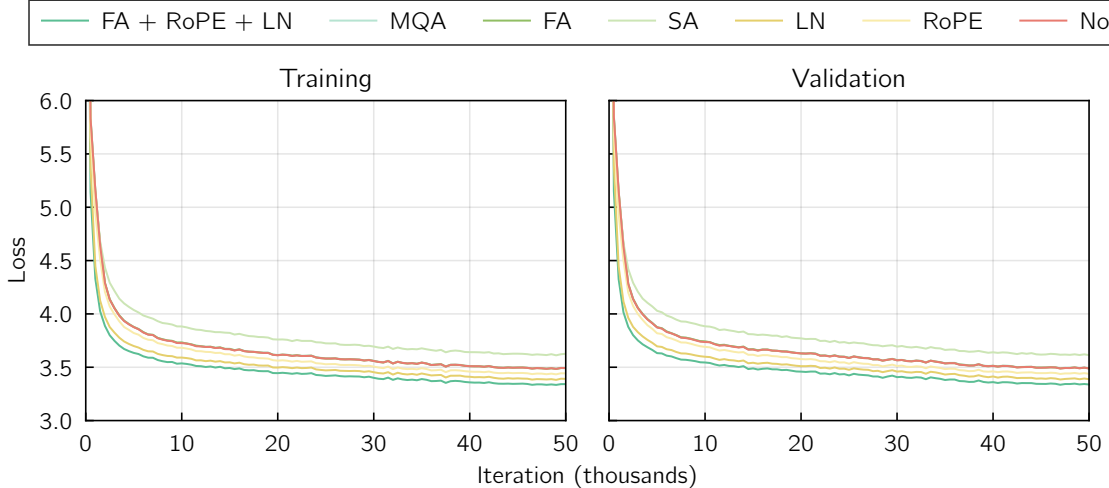


FIG. 1. Training and validation loss curves during GPT training experiments for different choices of LLM algorithms. We report the cross-entropy loss every 500 iterations. FA, FlashAttention; LN, LayerNorm; SA, sparse attention.

which did not fully allow it to take advantage of sparse attention’s benefits.

TABLE IV. Comparison of algorithmic enhancements with their compute-equivalent gains (CEG) and performance metrics during GPT training experiments

Algorithm	Train Loss	Validation Loss	Training Time (min)	Average MFU (%) ^a	Final performance gain	CEG at 25k steps	CEG at 50k steps
Baseline	3.49	3.49	330.01	11.3	—	—	—
LayerNorm	3.39	3.39	340.67	11.0	2.18	2.35	2.24
RoPE	3.44	3.44	355.34	10.5	1.32	1.42	1.54
FlashAttention	3.49	3.49	250.34	15.0	0.99	0.71	0.81
Sparse Attention							
MQA	3.49	3.49	329.17	11.4	0.98	0.91	— ^b
LayerNorm + RoPE + FlashAttention	3.34	3.34	289.51	12.9	3.17	3.14	3.03

IV. DISCUSSION

We now discuss our findings from above. Table II demonstrates that compute-independent algorithmic advances do exist, but the most impactful algorithmic advancements tend to be compute-dependent. These findings are generally confirmed by our experiment: the compute-dependent advances at best did nothing and at worse slightly degraded our small-scale model’s performance, whereas the compute-independent advances improved performance at all scales.

Additionally, our compute-dependent vs. compute-independent framework should serve as a useful distinction for future AI research. If, before testing, researchers hypothesize that an algorithmic advancement is compute-independent, initial experiments can be run at smaller scales to confirm its usefulness. On the other hand, if the advancement is hypothesized to be compute-dependent, researchers may be inclined to start with larger-scale experiments, which would likely yield more relevant results. We therefore hope that going forward, this framework can be used to speed up the empirical validation step of the research process.

There are several limitations to our analysis. First, we do not account for the role of datasets in AI advancements, which may be particularly relevant for cases when training on other LLM outputs has been suggested (e.g., DeepSeek). This contrasts with prior work, such as Epoch AI’s dataset-equivalent gain estimates. Additionally, we do not consider inference-time improvements or reinforcement learning techniques, such as chain-of-thought prompting, reinforcement learning from human feedback, or DeepSeeks group-relative policy optimization methods, which may further impact LLM capabilities. Another key limitation is our assumption that algorithmic advancements act independently; in reality, they often work synergistically. Our computational study did not test every possible algorithmic combination, though the superior performance of LayerNorm + RoPE + FlashAttention suggests the existence of some stacking effect. Similarly, our estimates rely on evaluating algorithms in isolation from state-of-the-art models, whereas modern models integrate multiple advancements (e.g., LayerNorm, RoPE, and MQA, often with modifications like RMSNorm or LongRoPE). Consequently, our estimates may underestimate the true impact of algorithmic progress. Furthermore, it is unclear whether recent AI advancements are representative of long-term trends. Epoch AI’s research indicates that much of the progress in frontier models has been driven by compute scaling,^{4,36} raising the possibility that, as data and hardware availability plateau, future research will prioritize model compression, distillation, or other efficiency-driven methods.^{37–40} Additionally, our study limits its focus to language modeling, but our framework could be extended to other domains, such as computer vision⁴¹ or biological sequence modeling. Though results may vary, we expect similar trends to hold across different AI subfields.⁴²

Finally, we hypothesize about the cause for some algorithmic advancements being compute-dependent, while others are compute-independent. Among the compute-dependent advancements we analyzed above, we observe that each advancement directly affects the attention mechanism. For example, the transformer architecture introduced the attention mechanism, and in MoE, the attention mechanism is “split” across multiple experts. On the other hand, the compute-independent advances tend to not affect the attention mechanism. (Even though FlashAttention’s name suggests it deals with the attention mechanism, its advancement deals with the mechanism’s interaction with hardware and does not change the mechanism itself.) Our findings therefore suggest that the attention mechanism is the primary compute consumer (which falls in line with its $O(n^2)$ scaling), and that improvements to the mechanism consequently yield the greatest compute-equivalent gains. This further suggests that improving the attention mechanism should be a focus of further research.

A. Implications for Policymakers

Hardware controls, while clearly impactful, are no silver bullet. Even if the strictest controls were enacted and enforced, this would not suffice to guarantee continued U.S. AI dominance. Past compute-independent algorithmic advances show that even at lower compute levels, models can be improved through algorithmic innovation. This further implies that, even if compute were to suddenly freeze, we could still improve models through compute-independent advances. We point to our discussion of DeepSeek in Section II A as a real-world example of this. However, because the majority of algorithmic progress has appeared to come from compute-dependent algorithmic advancements, hardware controls are far from a futile effort. (We again point to DeepSeek’s CEO’s comments on compute touched on in Section II A.) The biggest capability advances have come from compute-dependent advances, and we do not expect this to change.

To the extent that algorithmic advances that improve the performance of small models also improve the performance of larger models, we also expect research and development to be easier. If results from experiments on smaller models translate to larger models, it will take less compute, researcher hours, and time to iterate, and it will be easier to run many experiments in parallel. If the bar for humans to run experiments is lower, we might also expect the bar for AI agents to run experiments to be lower, and we thus expect AI agents to be especially capable of finding these compute-independent advances.

Though the lower bar means that less-resourced researchers can find compute-independent advances, this also means that better-resourced actors can find more. In this way, organizations which already have access to

lots of compute are counterintuitively better positioned to discover new compute-independent advances: they have the resources to search more comprehensively across the range of possible compute-independent advances, and can likely automate that search more easily than competitors with less computation power and with less-capable agents. Indeed, this approach sounds remarkably similar to the one described by Google’s Chief Scientist Jeff Dean during his February 2025 appearance on the Dwarkesh Podcast:

I think one thing people should be aware of is that the improvements from generation to generation of these models often are partially driven by hardware and larger scale, but equally and perhaps even more so driven by major algorithmic improvements and major changes in the model architecture, the training data mix, and so on, that really makes the model better per flop that is applied to the model, so I think that’s a good realization. Then I think if we have automated exploration of ideas, we’ll be able to vet a lot more ideas and bring them into the actual production training for next generations of these models.

That’s going to be really helpful because that’s sort of what we’re currently doing with a lot of brilliant machine learning researchers: looking at lots of ideas, winnowing ones that seem to work well at small scale, seeing if they work well at medium scale, bringing them into larger scale experiments, and then settling on adding a whole bunch of new and interesting things to the final model recipe. If we can do that 100 times faster through those machine learning researchers just gently steering a more automated search process, rather than hand-babysitting lots of experiments themselves, that’s going to be really, really good.⁴³

One key consequence of our findings for policymakers who are interested in the most-capable models, then, is that it will be more difficult to define the frontier with pre-training compute alone. Record-breaking quantities of pre-training compute remain sufficient for frontier capabilities, but are less necessary. In addition to model distillation—not discussed here—which let smaller models achieve performance comparable to larger models by training on their outputs, algorithmic improvements could allow models to increase performance without crossing a FLOP threshold.

We are not aware of any meaningful way to address this gap—attempts to monitor or shape algorithmic research are one option, but do not seem politically feasible, even if there did exist the regulatory capacity to do that monitoring. Evaluations may be feasible for particular narrow capabilities,⁴⁴ but also remain outside the Overton window and are poor measures of general capabilities.

V. CONCLUSION

In this paper, we introduced a novel framework for classifying algorithmic advancements as either compute-dependent or compute-independent, providing a clearer understanding of how LLM capabilities can progress even under hardware constraints. Our empirical validation confirmed that compute-independent innovations like Layer Normalization, RoPE, and FlashAttention yield meaningful performance gains (up to $3.5\times$ compute-equivalent gain) even in resource-constrained settings. On the other hand, the most impactful advancements remain compute-dependent and often attention-focused, suggesting that export controls may slow, but cannot fully prevent, AI progress. Our investigation focused on algorithmic improvements to pretraining, and future work should address advancements in other places, for example, by estimating the compute-equivalent gain from chain-of-thought prompting. Such estimates might give us a more holistic view of the role non-architectural factors play in the increasing performance of frontier models.

[1] T. Fist and E. Grunewald, *Preventing AI Chip Smuggling to China: A Working Paper*, Working Paper (Center for a New American Security, 2023) center for a New American Security Technology and National Security

Program.

- [2] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, Scaling laws for neural language models (2020), arXiv:2001.08361 [cs.LG].
- [3] J. Sevilla, T. Besiroglu, B. Cottier, J. You, E. Roldn, P. Villalobos, and E. Erdil, Can ai scaling continue through 2030? (2024), accessed: 2025-02-14.
- [4] J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos, Compute trends across three eras of machine learning, in *2022 International Joint Conference on Neural Networks (IJCNN)* (2022) pp. 1–8.
- [5] A. Ho, T. Besiroglu, E. Erdil, D. Owen, R. Rahman, Z. C. Guo, D. Atkinson, N. Thompson, and J. Sevilla, Algorithmic progress in language models (2024), arXiv:2403.05812 [cs].
- [6] D. Hernandez and T. Brown, Measuring the algorithmic efficiency of neural networks (2020).
- [7] L. Heim and L. Koessler, Training compute thresholds: Features and functions in ai regulation (2024), arXiv:2405.10799 [cs.CY].
- [8] DeepSeek-AI, A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Guo, D. Yang, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Zhang, H. Ding, H. Xin, H. Gao, H. Li, H. Qu, J. L. Cai, J. Liang, J. Guo, J. Ni, J. Li, J. Wang, J. Chen, J. Chen, J. Yuan, J. Qiu, J. Li, J. Song, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Xu, L. Xia, L. Zhao, L. Wang, L. Zhang, M. Li, M. Wang, M. Zhang, M. Zhang, M. Tang, M. Li, N. Tian, P. Huang, P. Wang, P. Zhang, Q. Wang, Q. Zhu, Q. Chen, Q. Du, R. J. Chen, R. L. Jin, R. Ge, R. Zhang, R. Pan, R. Wang, R. Xu, R. Zhang, R. Chen, S. S. Li, S. Lu, S. Zhou, S. Chen, S. Wu, S. Ye, S. Ye, S. Ma, S. Wang, S. Zhou, S. Yu, S. Zhou, S. Pan, T. Wang, T. Yun, T. Pei, T. Sun, W. L. Xiao, W. Zeng, W. Zhao, W. An, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, X. Q. Li, X. Jin, X. Wang, X. Bi, X. Liu, X. Wang, X. Shen, X. Chen, X. Zhang, X. Chen, X. Nie, X. Sun, X. Wang, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yu, X. Song, X. Shan, X. Zhou, X. Yang, X. Li, X. Su, X. Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Y. Zhang, Y. Xu, Y. Xu, Y. Huang, Y. Li, Y. Zhao, Y. Sun, Y. Li, Y. Wang, Y. Yu, Y. Zheng, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Tang, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Wu, Y. Ou, Y. Zhu, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Zha, Y. Xiong, Y. Ma, Y. Yan, Y. Luo, Y. You, Y. Liu, Y. Zhou, Z. F. Wu, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Huang, Z. Zhang, Z. Xie, Z. Zhang, Z. Hao, Z. Gou, Z. Ma, Z. Yan, Z. Shao, Z. Xu, Z. Wu, Z. Zhang, Z. Li, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Gao, and Z. Pan, DeepSeek-V3 Technical Report (2024), arXiv:2412.19437 [cs].
- [9] P. Schmid, O. Sanseviero, A. Bartolome, L. von Werra, D. Vila, V. Srivastav, M. Sun, and P. Cuenca, Llama 3.1 - 405B, 70B & 8B with multilinguality and long context, <https://huggingface.co/blog/llama31> (2025).
- [10] M. AI, Introducing Llama 3.1: Our most capable models to date, <https://ai.meta.com/blog/meta-llama-3-1/>.
- [11] D. Amodei, On DeepSeek and Export Controls (2025), accessed: March 13, 2025.
- [12] J. Schneider, Deepseek: The Quiet Giant Leading China’s AI Race, <https://www.chinatalk.media/p/deepseek-ceo-interview-with-chinas> (2024).
- [13] T. Davidson, J.-S. Denain, P. Villalobos, and G. Bas, AI capabilities can be significantly improved without expensive retraining (2023), arXiv:2312.07413 [cs].
- [14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention Is All You Need (2023), arXiv:1706.03762 [cs].
- [15] We acknowledge that there is a difference between full self-attention and causal self-attention in this regard, however both mechanisms exhibit the same asymptotic growth, so they can be treated identically for our purposes.
- [16] Although notably, self-attention scales in $O(n^2)$ time, which is a major focal point of many other algorithms in this paper.
- [17] J. Droppo and O. Elibol, Scaling Laws for Acoustic Models (2021), arXiv:2106.09488 [eess].
- [18] R. Child, S. Gray, A. Radford, and I. Sutskever, Generating Long Sequences with Sparse Transformers (2019), arXiv:1904.10509 [cs].
- [19] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer (2017), arXiv:1701.06538 [cs].
- [20] W. Fedus, B. Zoph, and N. Shazeer, Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity (2022), arXiv:2101.03961 [cs].
- [21] D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. K. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang, DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models (2024), arXiv:2401.06066 [cs].
- [22] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, Adaptive Mixtures of Local Experts, *Neural Computation* **3**, 79 (1991).
- [23] L. Wang, H. Gao, C. Zhao, X. Sun, and D. Dai, Auxiliary-Loss-Free Load Balancing Strategy for Mixture-of-Experts (2024), arXiv:2408.15664 [cs].
- [24] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding (2020), arXiv:2006.16668 [cs].
- [25] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu,

- B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, Llama 2: Open Foundation and Fine-Tuned Chat Models (2023), arXiv:2307.09288 [cs].
- [26] N. Shazeer, Fast Transformer Decoding: One Write-Head is All You Need (2019), arXiv:1911.02150 [cs].
- [27] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, RoFormer: Enhanced Transformer with Rotary Position Embedding (2023), arXiv:2104.09864 [cs].
- [28] Y. Ding, L. L. Zhang, C. Zhang, Y. Xu, N. Shang, J. Xu, F. Yang, and M. Yang, LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens (2024), arXiv:2402.13753 [cs].
- [29] O. Press, N. A. Smith, and M. Lewis, Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation (2022), arXiv:2108.12409 [cs].
- [30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2019), arXiv:1810.04805 [cs].
- [31] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness (2022), arXiv:2205.14135 [cs].
- [32] T. Dao, FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning (2023), arXiv:2307.08691 [cs].
- [33] J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao, FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision (2024), arXiv:2407.08608 [cs].
- [34] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, Mixed Precision Training (2018), arXiv:1710.03740 [cs].
- [35] J. L. Ba, J. R. Kiros, and G. E. Hinton, Layer normalization (2016), arXiv:1607.06450 [stat.ML].
- [36] J. Sevilla and E. Roldn, Training compute of frontier ai models grows by 4-5x per year (2024), accessed: 2025-03-03.
- [37] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, LLM.int8(): 8-bit matrix multiplication for transformers at scale, in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Nips '22 (Curran Associates Inc., Red Hook, NY, USA, 2022) pp. 30318–30332.
- [38] M. Fishman, B. Chmiel, R. Banner, and D. Soudry, Scaling FP8 training to trillion-token LLMs (2024), arXiv:2409.12517, arXiv:2409.12517 [cs].
- [39] H. Peng, K. Wu, Y. Wei, G. Zhao, Y. Yang, Z. Liu, Y. Xiong, Z. Yang, B. Ni, J. Hu, R. Li, M. Zhang, C. Li, J. Ning, R. Wang, Z. Zhang, S. Liu, J. Chau, H. Hu, and P. Cheng, FP8-LM: Training FP8 Large Language Models (2023), arXiv:2310.18313 [cs].
- [40] P. Micikevicius, D. Stolic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, and H. Wu, FP8 Formats for Deep Learning (2022), arXiv:2209.05433, arXiv:2209.05433 [cs].
- [41] E. Erdil and T. Besiroglu, Algorithmic progress in computer vision (2023), arXiv:2212.05153 [cs].
- [42] N. Maug, A. O’Gara, and T. Besiroglu, Biological sequence models in the context of the ai directives (2024), accessed: 2025-03-03.
- [43] D. Patel, Jeff dean & noam shazeer 25 years at google: from pagerank to agi, Podcast (2025), two of Gemini’s co-leads on Google’s path to AGI.
- [44] H. Karnofsky, A sketch of potential tripwire capabilities for ai (2024), carnegie California.