On the Origin of Algorithmic Progress in AI

Hans Gundlach [1] **Alex Fogelson** [1] **Jayson Lynch** [1] **Ana Trišović** [1] **Jonathan Rosenfeld** [1] **Anmol Sandhu** [2] **Neil Thompson** [1]

Abstract

Algorithms have been estimated to increase AI pretraining FLOP efficiency by a factor of $22,000$ between 2012 and 2023 (Ho et al., 2024). Running ablation experiments on key innovations from this time period, we are able to account for less than $10\times$ of these gains at small compute scales. Surveying the broader literature, we estimate that additional innovations not included in our ablations also account for less than $10\times$, yielding less than $100\times$ efficiency gains total in low-compute regimes. This leads us to conduct scaling experiments, which reveal that much of this efficiency gap can be explained by algorithms with scale-dependent efficiency improvements. In particular, we conduct scaling experiments between LSTMs and Transformers, finding exponent differences in their compute-optimal scaling law while finding little scaling difference for many other innovations. These experiments demonstrate that – contrary to standard assumptions – an algorithm's efficiency gains are tied to compute scale. Using experimental extrapolation and literature estimates, we account for $6,930\times$ efficiency gains over the same time period, with scale-dependent innovations accounting for close to 90% of log-efficiency gains by 2023. Our results indicate that algorithmic progress for small models has been far slower than previously assumed, and that measures of algorithmic efficiency are strongly reference-dependent.

1. Introduction

Progress in AI over the past decade has been driven by two tightly coupled forces: rapidly increasing compute budgets and a succession of algorithmic efficiency innovations in architectures, optimization, and other training practices. While compute growth is relatively straightforward to measure, we still lack a clear accounting of algorithmic progress—what concrete changes drive efficiency gains, how large those gains are, and whether they persist across compute scales.

Recent work by Ho et al. (2024) has started to address this question. By analyzing hundreds of historical language models, they estimate that in the last 10 years, algorithmic progress has contributed more than 4 orders of magnitude in compute efficiency, while raw compute scaling has increased 7 orders of magnitude based on analyzing historic AI literature. Specifically, models have gotten roughly $22,000\times$ more efficient across the sum total of innovations in algorithms from 2012-2023, supposedly allowing the same performance level with drastically fewer FLOPs.

However, a precise decomposition of this progress remains largely unknown, and many questions about the origins of algorithmic advancement remain underexplored. For instance, how do algorithmic advances interact? Are algorithmic advances driven by a series of small improvements, or a few large ones? Do algorithmic improvements follow smooth, Moore's law-like trends, or are they driven by punctuated equilibria? Answering these questions has important implications for the future of AI, as it could enhance the predictability of future algorithmic gains, guide innovation efforts towards specific areas of improvement, and inform our expected returns to compute scaling.

In order to answer these questions, we take three complementary approaches: we conduct ablation experiments on important algorithmic advances in language models; we conduct scaling experiments to measure differences in optimal scaling across architectures; and we perform theoretical analysis of transitions in data and parameter scalings. Our experiments focus primarily on architectural algorithmic modifications, but we try to address other methods to improve efficiency, like tokenizers, using literature estimates. Our analysis points to three core conclusions:

1. **We find most algorithmic innovations we experimentally evaluate have small, scale-invariant efficiency improvements** with less than $10\times$ compute efficiency gain overall, and representing about 10% of total improvements at 2023 compute levels. This suggests that scale-invariant algorithmic progress contributes

[1]MIT FutureTech, CSAIL, Cambridge, MA, USA [2]Olin College, Needham, MA, USA. Correspondence to: Hans Gundlach <hansgund@mit.edu>, Neil Thompson <neil_t@mit.edu>.

Second Workshop on Technical AI Governance Research (TAIGR) @ ICML 2026, Seoul, South Korea. 2026. Copyright 2026 by the author(s).

only a minor share of overall efficiency improvements (Section 3).

2. **We find two strongly scale-dependent algorithmic innovations: LSTMs to Transformers, and Kaplan to Chinchilla re-balancing.** Together, these account for 89% of total efficiency gains at 2023 compute levels. This implies that algorithmic progress for small-scale models is several orders of magnitude smaller than previously thought (Section 4).

3. **We show that in the presence of scale-dependent innovations, not only do efficiency gains require continued compute investment, but the rate of algorithmic progress strongly depends on your choice of reference algorithm.** In other words, the rate of progress in successive models can appear exponential relative to one baseline algorithm, yet be zero relative to another (Section 5.2).

Collectively, these findings suggest that gains in algorithmic progress may be inherently scale-dependent, requiring ever-increasing compute to realize their benefits, and implying that algorithmic progress has benefited model builders with greater compute resources more than smaller players in AI development.

1.1. Previous Work

A growing literature measures the contribution of algorithmic advances to ML efficiency. In computer vision, Hernandez & Brown (2020) found that the training FLOPs required to reach AlexNet-level performance fell by $44\times$ from 2012–2019. In language modeling, Ho et al. (2024) estimate roughly $22{,}000\times$ algorithmic efficiency gains from 2012–2023, though this magnitude has been questioned: Whitfill (2025) argues that correcting for observational bias (notably, correlations between compute scale and measured efficiency) may reduce estimated progress by about an order of magnitude. Experimental decompositions of historical gains remain limited; Sanderson et al. (2025) report modest improvements (e.g., $\sim 1.7\times$) from specific components such as rotary embeddings and layer normalization.

A central open question is whether algorithmic gains are scale-dependent. Several studies report little change in scaling exponents across architectures or optimizers (Hestness et al., 2017; Bansal et al., 2022), and theory work often emphasizes data properties as the primary driver of scaling behavior (Rosenfeld, 2021; Michaud et al., 2023; Bahri et al., 2024). Conversely, other empirical results suggest architecture can affect scaling: Droppo & Elibol (2021) find steeper scaling for Transformers than LSTMs in acoustic modeling, and Sanderson et al. (2025) theorize that major transitions (e.g., LSTM→Transformer) may have scale-dependent effects. Motivated by these mixed findings, we run controlled ablations and scaling experiments to identify which changes drive measured efficiency gains in language models, and how those gains vary with compute.

2. Compute-Equivalent Gains with Scale Dependence

To compare algorithmic choices in training efficiency, we use the compute-equivalent gains (CEG) framework (Davidson et al., 2023; Ho et al., 2024; Sanderson et al., 2025), and extend it to allow scale-dependent improvements. For two training algorithms A and A' (where "algorithm" includes architectural changes such as LSTM-to-Transformer, as well as training-recipe and component changes such as activation functions, normalization, positional encodings, learning-rate schedules, optimizers, and data–parameter allocation, etc.), define a *CEG function* $f(C)$ such that training with A' using compute $C/f(C)$ matches the performance of training with A using compute C. A *CEG multiplier* is then the point evaluation $f(C_0)$ for a fixed reference compute C_0 (equivalently, at a fixed performance level). This distinction matters in our setting because scale-dependent innovations can make CEG multipliers appear to grow over time even when the underlying comparison between algorithms is not constant across compute. Formal definitions and edge cases are given in Appendix H.

3. Scale-Invariant Algorithms

We estimate the compute-equivalent gains (CEGs) of several widely used training choices via controlled ablation experiments. Concretely, we vary one component at a time (activation functions, positional encodings, normalization, learning-rate schedules, and optimizers) and measure the compute required to reach a fixed validation threshold, yielding CEG multipliers (Section 2). We also test whether improvements compose multiplicatively by ablating multiple components jointly (Appendix I).

Across the compute regime of our experiments, these changes appear largely scale-invariant: in Section 4 we find near-identical scaling exponents between our baseline Transformer and a Transformer with all post-2017 components ablated (Figure 2). Overall, the cumulative scale-invariant gains we measure are modest (under $10\times$ in total), with most individual changes contributing less than $2\times$. This motivates our focus in Section 4 on scale-dependent innovations.

3.1. Ablation Experiments

We run ablations on a small Transformer baseline and compute CEG multipliers at a common validation threshold; full experimental details and per-component results are deferred to Appendix I. Figure 1 summarizes the measured multipli-

ers. In brief, we find small improvements from activation functions and positional encodings, negligible differences across common learning-rate schedules, and larger (but still $< 2\times$) gains from normalization and optimizer choice. We further analyze optimizer scaling separately in Appendix F.

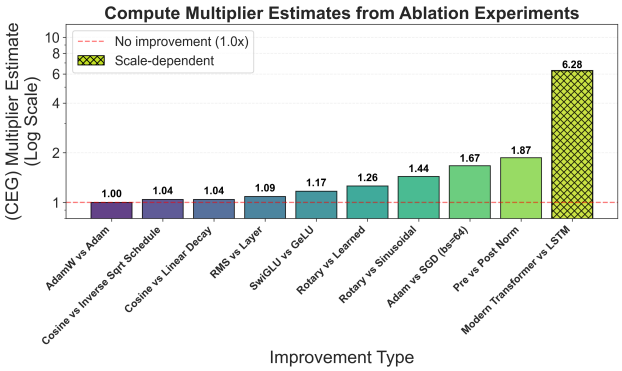

Figure 1. Compute-equivalent gain (CEG) multipliers from ablations on a *small* Transformer. Most component-level changes yield modest efficiency gains. Later experiments show only one ablation is scale-dependent.

3.2. Literature-Based Components

Some important contributors to efficiency are difficult to evaluate cleanly in our setup, namely mixture-of-experts (MoE) architectures, tokenizers, and innovations in data selection. When reporting efficiency gains over time, we judge literature estimates for tokenizers ($1.68\times$) and data ($\leq 3\times$) to be highly uncertain, and therefore only include MoE ($2\times$) in our stylizations and aggregations. Further discussion on literature estimates can be found in Section 6.1 and Appendix J.

3.3. Implication: Small, Skewed Scale-Invariant Gains

At small scales, the aggregate effect of the scale-invariant components we measure is limited, and the distribution is skewed: a handful of changes account for most of the gains while many contribute little. This motivates the scaling experiments in Section 4, where we test whether the largest historical gains arise from scale-dependent innovations rather than the accumulation of many scale-invariant tweaks.

4. Scale-Dependent Algorithms

Are algorithmic gains larger at larger compute scales? To answer this question, we conduct scaling experiments across architectures, optimizers, and the innovations from Section 3.1 to get a better sense of how their efficiency gains scale with compute. In addition, we exhibit a mathematical analysis of the Kaplan to Chinchilla scaling shift, which gives a particularly interesting example of a scale-dependent algorithmic change that is not an exponent shift.

Algorithm	Modern Innovations	Scaling policy
Modern Transformer	Included	Compute Optimal
Retro Transformer	Excluded	Compute Optimal
Kaplan Transformer	Excluded	Kaplan scaling
LSTM baseline	N/A	Compute Optimal

Table 1. Model configurations used in scaling experiments. "Kaplan Transformer" denotes the same Retro model evaluated under Kaplan-style parameter–data allocation (Appendix C).

4.1. Scaling Experiments: LSTM to Transformer

4.1.1. Experimental Setup for Scaling

We compare three training algorithms: a **Modern Transformer**, a **Retro Transformer** that reverts key post-2017 components (Section H.1), and a **pre-Transformer LSTM baseline**. To separate *architecture* from *scaling policy*, we also define a **Kaplan Transformer** as the Retro Transformer evaluated under Kaplan-style parameter–data allocation via an analytical adjustment (Appendix C). Table 1 summarizes the configurations; full architecture and hyperparameter details are deferred to Appendix C.

Transformer baselines. Our Modern Transformer is a small, vanilla decoder-only Transformer with standard contemporary choices (e.g., rotary position encodings, RMSNorm prenorm, and SwiGLU), using a fixed width-to-depth aspect ratio appropriate for the small-model regime. The **Retro Transformer** matches the Modern Transformer except that we revert four components (activation, normalization placement/type, positional encoding, and learning-rate schedule) to approximate an early Transformer-style recipe. We keep these definitions fixed across scales and tune learning rates uniformly across model families; exact architectural choices and all hyperparameters are listed in Appendix C.

LSTM baseline We use a vanilla LSTM intended to reflect strong pre-Transformer practice (following Melis et al. (2017)) and exclude later variants (e.g., GRUs, Mogrifier, xLSTM). We use a **single LSTM layer** because (i) Melis et al. (2017) finds 1-layer LSTMs are competitive at $\sim$10M parameters, and (ii) in our scaling runs, 1-layer is consistently better than 2-layer at both the smallest and largest sizes we test, aligning with the aspect-ratio guidance in Droppo & Elibol (2021). We adopt standard LSTM initialization and regularization conventions from Melis et al. (2017) (Table 4). Because our training regime uses large-scale web data where overfitting is minimal, we set LSTM dropout to zero (analogous to common Transformer practice in this setting (Hoffmann et al., 2022)); this materially

improves validation loss in our compute-optimal runs (Appendix C).

Scale range and tuning. We scale each family by varying hidden size over a fixed grid (Appendix A), yielding Transformers spanning ∼1.6M–29M parameters and LSTMs spanning ∼1.6M–11.2M parameters. For each family, we run matched learning-rate tuning and fit a smooth function from sizes to learning rates across scales (Appendix A.1).

We scale our default Modern Transformer and Retro Transformer, with algorithmic changes ablated. For each setup, we follow conventional scaling practices, keeping depth-to-width ratio constant.[1]

4.1.2. Transformers are a Scale-Dependent Improvement to LSTMs

We plot the difference in scaling between LSTMs and Modern Transformers in Figure 2A and between Modern and Retro Transformers in Figure 2B. For each, we subtract the irreducible loss component (1.9, see Appendix G) and fit a power law to all Pareto-optimal (frontier) points from 10^{16} FLOPs onward.[2]

Our scaling graphs suggest that improvements in neural network architecture are not scale invariant and have increasing returns to scale. Interestingly, using an irreducible loss adjusted threshold of 3.4, our LSTM model exhibits $6.3\times$ less compute-efficiency than our transformer model, while a threshold of 2.5 yields an efficiency gap of $26\times$ (Figure 2). At 2017 compute levels, the efficiency gain is approximately $91\times$, slightly larger than previous estimates from the literature at these scales. Ho et al. (2024) attributes a $60\times$ gain to this transition. Sanderson et al. (2025) estimates that this transition had algorithmic FLOP efficiency gains of $20 - 50\times$ at scales of around 10^{19} FLOPs.

Meanwhile, the choice between Adam and SGD, as well as the difference between a Modern and a Retro Transformer, seems to be scale-invariant. This provides direct evidence that optimizers may exhibit constant CEG functions, and strong evidence that our ablated algorithms would not individually show scale-dependencies.

4.2. Transition from Kaplan to Chinchilla Scaling Laws

In our discussion so far, we have focused on model training choices and have paid relatively little attention to how models are scaled, assuming they are scaled optimally between dataset size and parameter count for a set of algorithms. However, the shift from so-called "Kaplan" to "Chinchilla" scaling practices are a particularly interesting example of scale-dependent algorithmic progress that is not an exponent shift.

Kaplan et al. (2020) advocates for a smaller level of data scaling compared to Hoffmann et al. (2022). Subsequent literature found that Kaplan misestimated scaling law exponents due to small sizes, not accounting for embedding parameters, and using a constant warmup size (Porian et al., 2024; Pearce & Song, 2024). In contrast, Hoffmann et al. (2022) accounted for all parameters and found that data and parameters should be scaled equally while scaling compute (so-called "Chinchilla-scaling"). Our analysis provides a closed form for the CEG function between Kaplan and Chinchilla scaling, and displays larger gains than previous estimates (Ho et al., 2024).

To formalize this switch from Kaplan to Chinchilla laws, we take Chinchilla-optimal scaling laws as ground truth, and assume model trainers allocate parameters and data according to misspecified Kaplan scaling laws (see Appendix E). This form of algorithmic improvement has clear scale-dependent effects, though not a simple exponent change. Training efficiency initially converges between the two scaling laws (within the regime tested by Kaplan), but diverges at larger scales (Figure 7). However, the Chinchilla training efficiency gains remain below 10x for much of the scale used in contemporary machine learning.

5. Algorithmic Progress Depends Strongly on Compute, Reference Points

Our theoretical and experimental results allow us to decompose algorithmic progress into individual innovations introduced over time. This leads to two main observations. First, many measured CEG gains may be a consequence of compute scaling rather than a multitude of algorithmic innovations. Second, when innovations are scale dependent, the growth rates of CEG multipliers depends entirely on the chosen reference model, meaning that progress may appear unbounded with respect to one algorithm while non-existent with respect to another.

5.1. Compute (Not Time) May Explain Most Algorithmic Progress

Existing estimates of algorithmic progress measure the rate of progress in algorithms as a function of time (Ho et al., 2024). However, our analysis opens the question as to whether this time-dependence is instead driven by steady increases in compute investment. Estimates show the compute budgets of notable models have exponentially increased at a rate of $4.1\times$ per year (Sevilla & Roldán, 2024). Thus as compute budgets increase exponentially, the rate of algorithmic progress may in fact be driven more by the regularity

[1] See Appendix A and the following discussion for more details on our experimental setup.

[2] Additional scaling experiments for Adam-to-SGD are described in Appendix F.

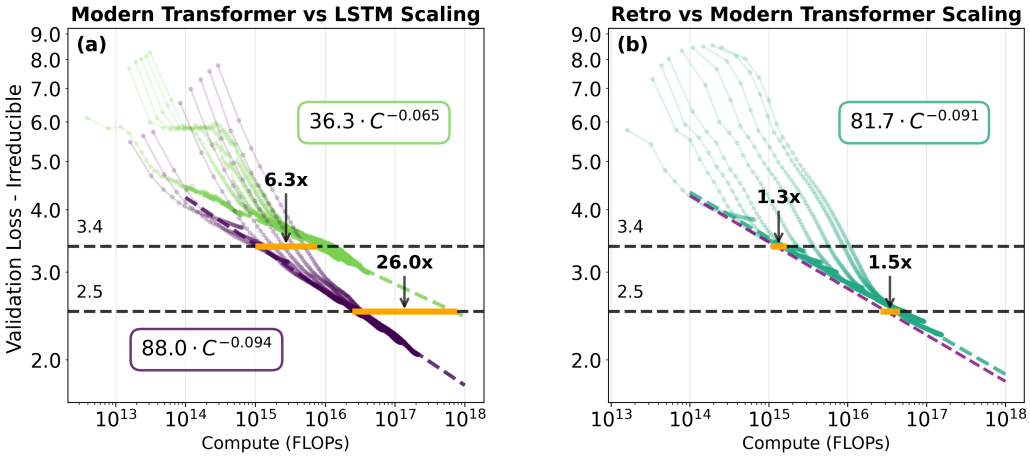

Figure 2. Figure (a) depicts the scaling difference between a Modern Transformer in purple and a standard LSTM in green. Figure (b) depicts the scaling difference between a Modern Transformer in purple and a Retro Transformer in blue, where all post-2017 innovations are ablated. LSTMs seem to have significantly different scaling exponents, while post-2017 transformers have minimal effect on scaling. All graphs depict the training curve for models with hidden dimensions between 32 and 256 with all other hyperparameter scaled proportionately.

of compute scaling rather than the discovery of new innovations.

We stack our empirical results, as well as literature estimates for mixture of experts and theoretical results, to extrapolate the expected CEG gains over varying compute scales (with respect to LSTMs). To compare our results to literature estimates (which track gains over time), we use the (aforementioned) exponential relationship between time and compute among notable models. We denote this trend as the "compute frontier" (distinct from the steeper trajectory one obtains by considering only the top-performing model at each date). This allows us to calculate the cumulative CEG function over historically relevant time intervals.

When stacking our ablated, scale-invariant innovations, we scale down their total CEG multiplier according to our interaction experiments (Section H.1). When reporting the decomposition, the results remain log-proportional to the independent multipliers. Moreover, when reporting gains from Kaplan Transformers, we analytically adjust our Retro Transformer according to gains from the Kaplan-to-Chinchilla CEG function.

Importantly, between 2017 and 2023, we find the overwhelming majority of algorithmic progress can be accounted for by two scale-dependent innovations: the switch from LSTM to Kaplan Transformers and the rebalancing to Chinchilla scaling. Of the total measured progress we account for $6,930\times$ (relative to an LSTM), we find that $726\times$ is achieved through LSTMs to Kaplan Transformers, and $3.7\times$ is attributable to Chinchilla rebalancing. Together, these comprise 89% of total relative efficiency gains, the vast majority of our measured progress. These results are stylized in Figure 3.

Though our experiments do not claim to be exhaustive, we compare our findings with estimates from the literature. Namely, between 2012 to 2023, Ho et al. (2024) found a doubling time of 8 months, or $2.83\times$ per year, for a total efficiency gain of $22,000\times$. In contrast, the growth rate of our CEG multiplier is approximately $2.23\times$ annually over the same time period, for a total of $6,930\times$, of which $2,700\times$ (89%) is due to scale-dependent changes. This leaves a gap of $3.18\times$ from our estimates, which could be from data selection, tokenizer advancements, or a long tail of innovations not captured in our analysis. We include a sensitivity analysis of our results in Appendix B.

5.2. CEG Multipliers Depend Strongly on Reference Algorithms

Our findings decomposing algorithmic progress suggest that scale-dependent changes may have disproportionate effects on the overall measured progress, contributing orders of magnitude more than their scale-invariant counterparts. *However, a deeper problem exists once scale-dependent algorithms are introduced.*

Consider a sequence of Modern Transformer models $M_1, M_2, ..., M_t$, each enhanced with a mixture-of-experts architecture, and trained with exponentially increasing compute. If we want to measure the growth rate of CEG multipliers (i.e., the rate of algorithmic progress), we need to choose a reference algorithm, such as baseline LSTMs or dense transformers. Measuring the CEG multiplier with respect to LSTMs for each model M_t yields an exponential in t, with a growth rate of about 63% annually (Figure 4).

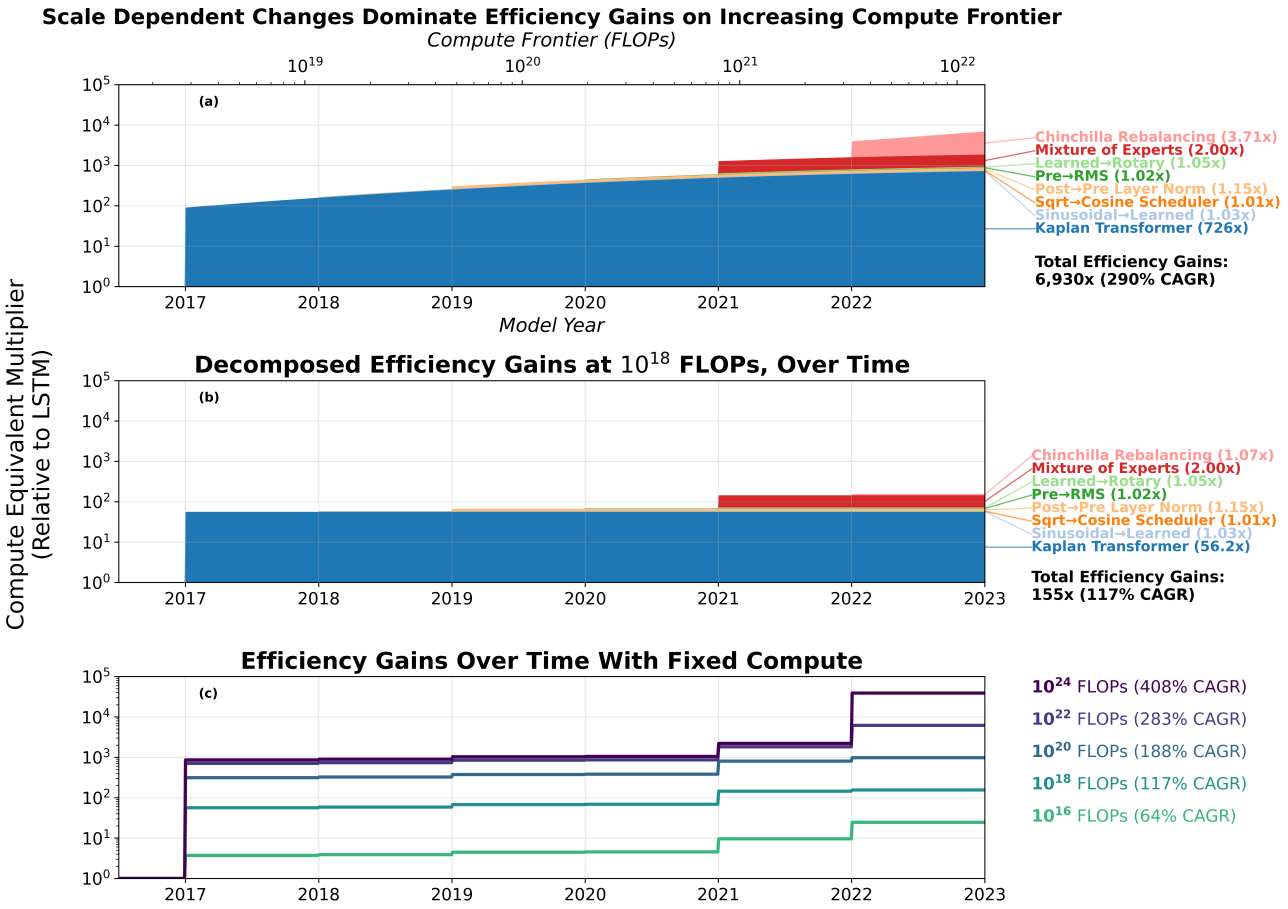

Figure 3. LSTM to Transformer transition has been responsible for most of the algorithmic progress out of the innovations we review. We adjust scale-invariant transitions to be log-proportional to our overall ablation results, given the interaction effects we demonstrate in Section H.1.

However, measuring with respect to the dense Transformers yields a constant multiplier of $2\times$ (see Section J for literature estimates), and therefore a 0% growth rate. Thus choosing one reference point yields an *exponential growth* rate in algorithmic efficiency, while a different reference point yields *zero growth*. Figure 4 also shows the apparent growth in CEG multiplier from the Kaplan-Chinchilla transition, showing that these innovations can even appear to have variable growth rates over compute scales.

Only once scale-dependent algorithms are considered, selecting different reference algorithms for computing CEG multipliers can determine whether growth appears exponential, non-existent, or somewhere in between. In this sense, algorithmic progress with scale-dependent innovations is strongly contextual: even if you are completely certain you will measure exponential growth rates in the future with respect to some *fixed algorithm*, this offers no guarantee of any progress at all *relative to current models*. Importantly, this *is not* merely the claim that "we cannot be assured algorithmic progress will continue." Rather it is a statement about the measurement framework: **one can continue to *measure* arbitrarily large rates of progress from some fixed point while realizing none of these gains in practice.**

6. Limitations and Further Work

6.1. Impacts of Data Selection

We do not do any dataset ablation experiments. In general, it is hard to compare performance between datasets as perplexity is a dataset-relative benchmark. However, it is generally thought that datasets play an important role in determining neural scaling law exponents (Rosenfeld, 2021; Michaud et al., 2023). Sorscher et al. (2022) found that neural scaling laws can even be exponential rather than power laws under the right data pruning metric. Nevertheless, in practice, while a better dataset may improve power laws, the difference between most datasets on evaluation metrics is moderate. Li et al. (2024) did a large-scale study comparing pretraining efficiency on new and historical datasets.

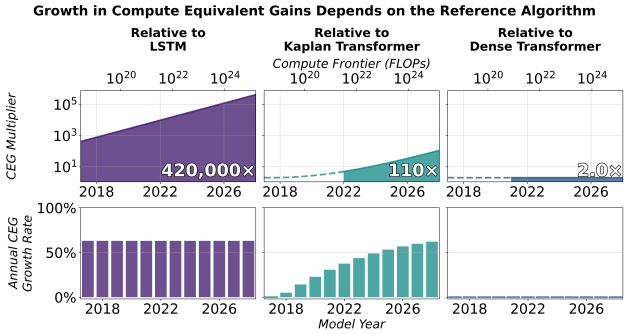

Figure 4. Top row depicts the overall CEG gain of a MoE transformer with respect to LSTMs, Kaplan transformers, and dense transformers. The bottom row depicts CEG multiplier growth rates with respect to the same algorithms, extrapolated to 2028. Depending on the choice, growth rates vary dramatically. Years are aligned with approximate historical model sizes.

They claim a $10\times$ efficiency gain for their DCLM baseline dataset, but most historical web datasets are within a factor of $3\times$ FLOP efficiency of one another in their study at scales of 10^{20} to 10^{22} FLOPs.

6.2. Our Experiments are Relatively Small and Non-Exhaustive

Our experiments are conducted at small scales compared to more recent scaling studies (i.e., Hoffmann et al. (2022)). However, our studies are done at slightly lower but similar scales to papers that originally explored many of these innovations, like Melis et al. (2017).

Finally, there are many things we did not test experimentally. These include mixture of experts, tokenizers, improvements in data, and other innovations which may be underrepresented in the literature or less widely known. These are important contributors to algorithmic progress, but we are only able to give literature-based estimates. We also do not test innovations involved in the creation of LSTMs. However, many LSTM variations have little benefit (Greff et al., 2016), and we use a generic architecture that would have been available in 2012. Further, we did not run scaling interventions on all the algorithms in our study individually, though when we ablate all algorithmic improvements, we find very small exponent differences in scaling exponents. We conclude from this that individual innovations admit negligible exponent shifts alone.

6.3. The Burden of Optimality

Another key limitation is the fickle nature of hyperparameters. For instance, we found it nearly impossible to scale a transformer trained with SGD and a batch size over 128. We are more confident in our results because our performance and exponents aligns well with other studies like Porian et al. (2024) and Melis et al. (2017). Notably, Porian et al. (2024) conducts chinchilla optimal scaling and finds scaling exponents from .092 to .106. Using the same tokenizer but slightly different data than Porian et al. (2024), we achieve a scaling exponent of .094. Hoffmann et al. (2022) achieves a scaling exponent of .154 but uses both a different tokenizer and different data.

However, our study leaves open the possibility of "golden" hyperparameters that could significantly lower the gap between the algorithmic improvements we measure or even change the experimental scaling relationship. For instance, SGD was long thought to be inherently inferior to Adam for transformer training, but studies like Srećković et al. (2025) demonstrate a very small performance gap with the right hyperparameters. We found very little literature on scaling LSTMs. Therefore, we could find improved scaling for LSTMs if extensive hyperparameter sweeps were done at all scales. This becomes computationally infeasible with much larger models, as determining optimal scaling for transformers took many years of progress in the machine learning community. Finding such an optimal scaling for other architectures and settings might require similar effort.

6.4. Beyond FLOP Efficiency: Reasoning and Other Dimensions of Progress

Reasoning-optimized language models have recently emerged as a new pathway for improving capabilities, with developers increasingly optimizing models to "think longer." Ho & Berg (2025) estimate that the shift from conventional to reasoning models is equivalent to a $10\times$ gain in pretraining compute, though these gains come with substantially higher inference costs and fall outside the scope of our study.

More broadly, our focus on training FLOP efficiency necessarily omits many critical innovations. Several changes we test (e.g., RMSNorm and learning-rate schedules) show modest FLOP-efficiency gains but materially improve stability, convergence reliability, and trainability. Other advances reduce wall-clock time and energy without changing FLOP counts (e.g., Flash Attention), while capability-enabling methods such as instruction fine-tuning (Wei et al., 2021) and constitutional AI change model utility without improving pretraining efficiency. Finally, systems advances (e.g., pipeline and tensor parallelism) make large-scale training more feasible (Barnett, 2025). Taken together, these examples suggest that algorithmic progress is only partially captured by scalar FLOP-efficiency metrics, and is better understood along multiple axes including efficiency, stability, capability, and deployment cost.

7. Discussion

Our experimental and theoretical analysis reveals that algorithmic progress in language models exhibits fundamentally different behavior across compute scales, with implications for understanding historical progress and future trajectories.

7.1. Reconciling Small-Scale Measurements with Large-Scale Progress

At performance thresholds comparable to Transformers trained with 10^{15} FLOPs (i.e., NLL = 5.3), we measure total algorithmic efficiency gains of approximately $6.28\times$ relative to baseline LSTMs when accounting for all tested innovations. Combined with literature estimates – MoE estimated at ($\sim 2\times$), improved datasets at ($\sim 3\times$), and tokenizers at ($\sim 1.6\times$) – this totals $60.3\times$, well under $100\times$. This contrasts sharply with Ho et al. (2024)'s estimate of $22,000$ efficiency gains the last decade, leaving an apparent gap of $350\times$ or more.

However, this discrepancy resolves when accounting for scale-dependent innovations. Extrapolating our measured scaling exponents to 2023 compute budgets (1.3×10^{22} FLOPs), the LSTM to Kaplan Transformer transition alone accounts for $726\times$ of the efficiency gains. Combined with the Chinchilla rebalancing gains of $3.7\times$ and $2.6\times$ for scale-invariant innovations, we estimate total efficiency gains of approximately $6,930\times$, substantially closing the gap with literature estimates over the same time period.

Algorithmic Versus Hardware Improvements Algorithmic progress is traditionally seen as much faster than hardware progress. For instance, Ho et al. (2024) estimates algorithmic progress of $22,000$ over a ten year period while Moore's law over a ten year period would only amount to efficiency gains of $32\times$. However, at small scales, for instance using 10^{18} FLOPs, algorithmic progress is only $20\times$ – less than hardware progress (See Figure 3).

7.2. The Concentration of Progress in Architectural Transitions

Our decomposition reveals that algorithmic progress exhibits extreme concentration: the LSTM-to-Transformer architectural shift comprises 74% of measured efficiency gains at 2023 compute levels, with the Kaplan-to-Chinchilla rebalancing accounting for most of the remainder. The distribution of scale-invariant innovations is highly skewed, with a handful of improvements approaching or exceeding $2\times$, while most innovations contribute less than $1.5\times$.

This concentration has two important implications. First, algorithmic progress appears more punctuated than smooth exponential trends would suggest, with long periods of incremental refinement interrupted by rare architectural transitions. Second, the small magnitude of most scale-invariant improvements ($< 10\times$ cumulatively) suggests that future progress may similarly depend on discovering fundamentally new architectures rather than incremental refinements of existing ones.

7.3. Implications for the Future of AI

Many forecasts of AI capabilities assume smooth exponential gains in training efficiency over time (Kokotajlo et al., 2025; Erdil et al., 2025) regardless of compute level. Our results suggest three critical corrections to this view.

Larger Players Gain More From Algorithms If efficiency improvements are scale-dependent, then algorithmic progress have not been uniform across small and large algorithmic providers. Model trainers with a large computational budget see much greater improvements with the introduction of new algorithms than smaller builders. Model builders scaling along the compute frontier see gains in the tens of thousands, while smaller model builders see algorithmic progress at levels much more similar to hardware progress.

Reference-dependence of Progress Measurements. In settings with scale-dependent algorithms, the apparent rate of "algorithmic progress" is not intrinsic—it depends on the reference baseline. The same sequence of frontier MoE Transformers can show rapid CEG-multiplier growth when measured against LSTMs (e.g., $\sim 63\%$/year) yet appear flat when measured against dense Transformers. Thus, meaningful efficiency trends require specifying a fixed reference algorithm; time-series estimates anchored to older baselines may overstate progress relative to contemporary models, even at the frontier.

Difficulty of Early Identification Scale-dependent innovations may appear modest at small scales, making it difficult to identify transformative improvements before committing substantial compute resources. The LSTM-to-Transformer transition showed only $6\times$ gains at 10^{15} FLOPs but more than $100\times$ gains at 10^{23} FLOPs. This suggests that limits to compute scaling pose obstacles not only to realizing efficiency gains but also to discovering them. This opens the possibility of algorithms that decrease performance at small scales, but which would improve performance if we scaled. Therefore, researchers must examine the scale-dependent nature of their improvements to identify the real impact of their innovations.

Will Algorithmic Progress Slow Down? These findings have important implications for society and AI governance. If efficiency gains primarily arise from scale-dependent innovations, then limits to compute scaling—whether from energy constraints, semiconductor supply chains, or regu-

latory restrictions—may substantially slow AI algorithmic progress. In addition, if algorithmic progress cannot be separated from compute investment, then the current exponential growth in AI capabilities may be more brittle than commonly assumed, requiring sustained increases in both computational resources and algorithmic breakthroughs.

Impact Statement

We study how measured training-efficiency gains in language models depend on compute scale and on the choice of reference algorithm. Our findings suggest that some prominent efficiency gains are strongly scale-dependent, so single-number estimates of "algorithmic progress" can be misleading without specifying both scale and baseline.

This work may improve forecasting and scientific communication by encouraging more careful reporting of efficiency comparisons. However, the same insights could be used to better target compute toward frontier-performing approaches, potentially accelerating capabilities and increasing inequality between "compute-available" and "compute-constrained" actors. Our empirical results are limited to relatively small-scale experiments and primarily to language-model loss; extrapolations to frontier systems and to downstream capabilities should be treated cautiously.

LLM Usage Statement

Large language models (LLMs) were used as general-purpose assistive tools during this project. Specifically, LLMs were used to assist with coding (writing and debugging experiment and analysis scripts), editing and formatting the manuscript (including LaTeX), and literature search. The authors reviewed and verified all LLM-assisted outputs, and take full responsibility for the content of the paper.

References

Ahn, K., Cheng, X., Song, M., Yun, C., Jadbabaie, A., and Sra, S. Linear attention is (maybe) all you need (to understand transformer optimization). *arXiv preprint arXiv:2310.01082*, 2023.

Ali, M., Fromm, M., Thellmann, K., Rutmann, R., Lübbering, M., Leveling, J., Klug, K., Ebert, J., Doll, N., Buschhoff, J., et al. Tokenizer choice for llm training: Negligible or crucial? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 3907–3924, 2024.

Bahri, Y., Dyer, E., Kaplan, J., Lee, J., and Sharma, U. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27), June 2024. ISSN 1091-6490. doi: 10.1073/pnas.2311878121. URL http://dx.doi.org/10.1073/pnas.2311878121.

Bansal, Y., Ghorbani, B., Garg, A., Zhang, B., Cherry, C., Neyshabur, B., and Firat, O. Data scaling laws in nmt: The effect of noise and architecture. In *International Conference on Machine Learning*, pp. 1466–1482. PMLR, 2022.

Barnett, P. Compute requirements for algorithmic innovation in frontier ai models. *arXiv preprint arXiv:2507.10618*, 2025.

Besiroglu, T., Erdil, E., Barnett, M., and You, J. Chinchilla scaling: A replication attempt. *arXiv preprint arXiv:2404.10102*, 2024.

Biderman, S., Black, S., Foster, C., Gao, L., Hallahan, E., He, H., Wang, B., and Wang, P. Rotary embeddings: A relative revolution. blog.eleuther.ai/, 2021. [Online; Accessed: 2025-11-25].

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

Davidson, T., Denain, J.-S., Villalobos, P., and Bas, G. Ai capabilities can be significantly improved without expensive retraining, 2023. URL https://arxiv.org/abs/2312.07413.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Droppo, J. and Elibol, O. Scaling laws for acoustic models. *arXiv preprint arXiv:2106.09488*, 2021.

Erdil, E. and Besiroglu, T. Algorithmic progress in computer vision. *arXiv preprint arXiv:2212.05153*, 2022.

Erdil, E., Potlogea, A., Besiroglu, T., Roldan, E., Ho, A., Sevilla, J., Barnett, M., Vrzla, M., and Sandler, R. Gate: An integrated assessment model for ai automation. *arXiv preprint arXiv:2503.04941*, 2025.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28 (10):2222–2232, 2016.

He, X. O. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*, 2024.

Hernandez, D. and Brown, T. B. Measuring the algorithmic efficiency of neural networks. *arXiv preprint arXiv:2005.04305*, 2020.

Hestness, J., Narang, S., Ardalani, N., Diamos, G., Jun, H., Kianinejad, H., Patwary, M. M. A., Yang, Y., and Zhou, Y. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

Ho, A. and Berg, A. Quantifying the algorithmic improvement from reasoning models, 2025. URL https://epoch.ai/gradient-updates/quantifying-the-algorithmic-improvement-from-reas Accessed: 2025-11-17.

Ho, A., Besiroglu, T., Erdil, E., Owen, D., Rahman, R., Guo, Z. C., Atkinson, D., Thompson, N., and Sevilla, J. Algorithmic progress in language models. *Advances in Neural Information Processing Systems*, 37:58245–58283, 2024.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models, 2022. URL https://arxiv.org/abs/2203.15556.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Keskar, N. S. and Socher, R. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017.

Kokotajlo, D., Alexander, S., Larsen, T., Lifland, E., and Dean, R. AI 2027: A scenario forecasting the impact of

superhuman ai over the next decade. Website, 2025. URL https://ai-2027.com/. Published April 3, 2025.

Li, H., Lo, K. M., Wang, Z., Wang, Z., Zheng, W., Zhou, S., Zhang, X., and Jiang, D. Can mixture-of-experts surpass dense llms under strictly equal resources? *arXiv preprint arXiv:2506.12119*, 2025.

Li, J., Fang, A., Smyrnis, G., Ivgi, M., Jordan, M., Gadre, S. Y., Bansal, H., Guha, E., Keh, S. S., Arora, K., et al. Datacomp-lm: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Michaud, E., Liu, Z., Girit, U., and Tegmark, M. The quantization model of neural scaling. *Advances in Neural Information Processing Systems*, 36:28699–28722, 2023.

Muennighoff, N., Rush, A., Barak, B., Le Scao, T., Tazi, N., Piktus, A., Pyysalo, S., Wolf, T., and Raffel, C. A. Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, 36:50358–50376, 2023.

Muennighoff, N., Soldaini, L., Groeneveld, D., Lo, K., Morrison, J., Min, S., Shi, W., Walsh, P., Tafjord, O., Lambert, N., et al. Olmoe: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

Pearce, T. and Song, J. Reconciling kaplan and chinchilla scaling laws. *arXiv preprint arXiv:2406.12907*, 2024.

Porian, T., Wortsman, M., Jitsev, J., Schmidt, L., and Carmon, Y. Resolving discrepancies in compute-optimal scaling of language models. *Advances in Neural Information Processing Systems*, 37:100535–100570, 2024.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21 (140):1–67, 2020.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34: 8583–8595, 2021.

Rosenfeld, J. S. Scaling laws for deep learning. *arXiv preprint arXiv:2108.07686*, 2021.

Sanderson, J., Foley, T., Guo, S., Qu, A., and Josephson, H. Rethinking llm advancement: Compute-dependent and independent paths to progress. *arXiv preprint arXiv:2505.04075*, 2025.

Sevilla, J. and Roldán, E. Training compute of frontier ai models grows by 4-5x per year, 2024. URL https://epoch.ai/blog/training-compute-of-frontier-ai-models-grows-by-4 Accessed: 2026-01-26.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Sorscher, B., Geirhos, R., Shekhar, S., Ganguli, S., and Morcos, A. Beyond neural scaling laws: beating power law scaling via data pruning. *Advances in Neural Information Processing Systems*, 35:19523–19536, 2022.

Srećković, T., Geiping, J., and Orvieto, A. Is your batch size the problem? revisiting the adam-sgd gap in language modeling. *arXiv preprint arXiv:2506.12543*, 2025.

Stanford University, D. o. C. S. Lecture 3: Architecture. https://github.com/stanford-cs336/spring2025-lectures/blob/e9cb2488fdb53ea37f0e38924ec3a170192cef3/nonexecutable/2025%20Lecture%203%20-%20architecture.pdf, March 2025. URL https://stanford-cs336.github.io/spring2025/index.html#coursework. Lecture slides for course CS 336 (Spring 2025).

Tay, Y., Dehghani, M., Abnar, S., Chung, H., Fedus, W., Rao, J., Narang, S., Tran, V., Yogatama, D., and Metzler, D. Scaling laws vs model architectures: How does inductive bias influence scaling? In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 12342–12364, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wei, J., Bosma, M., Zhao, V. Y., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Whitfill, P. Note on selection bias in observational estimates of algorithmic progress. *arXiv preprint arXiv:2508.11033*, 2025.

Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., and Liu, T. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

Table 2. Base hyperparameters used to train the Modern Transformer model. Taken from Porian et al. (2024) and optimal recommendations from Stanford University (2025).

Hyperparameter	Value
Width/Depth aspect ratio	16
Norm type	RMSNorm prenorm
Feedforward / model dimension	4
Activation function	SwiGLU
Positional encoding	Rotary
Sequence length	128
Stride	128
Batch size	64
Dropout	0.0
Weight decay	0.01
Gradient clipping	1.0
Learning rate schedule	cosine annealing
Min learning rate	$0.1\times$ max lr
Warmup fraction	10%
Default optimizer	AdamW
β_1	0.9
β_2	0.999
Initialization	GPT-2/BERT init
Embedding weight tying	True
Tokenizer	GPT-2 Tokenizer
Vocabulary	50257
Dataset	C4
Validation set size	500k tokens
Token-to-parameter ratio	40

Table 3. Architecture and hyperparameters for our Retro Transformer. This matches Table 2 with key algorithmic choices reverted.

Hyperparameter	Value
Width/Depth aspect ratio	16
Norm type	layer postnorm
Feedforward / model dimension	4
Activation function	GeLU
Positional encoding	Sinusoidal
Sequence length	128
Stride	128
Batch size	64
Dropout	0.0
Weight decay	0.01
Gradient clipping	1.0
Learning rate schedule	inverse square root decay
Min learning rate	$0.1\times$ max lr
Warmup fraction	10%
Default optimizer	AdamW
β_1	0.9
β_2	0.999
Initialization	GPT-2/BERT init
Embedding weight tying	True
Tokenizer	GPT-2 Tokenizer
Vocabulary	50257
Dataset	C4
Validation set size	500k tokens
Token-to-parameter ratio	40

Zhang, B. and Sennrich, R. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Zhang, Y., Chen, C., Ding, T., Li, Z., Sun, R., and Luo, Z. Why transformers need adam: A hessian perspective. *Advances in neural information processing systems*, 37: 131786–131823, 2024.

Zhao, R., Morwani, D., Brandfonbrener, D., Vyas, N., and Kakade, S. Deconstructing what makes a good optimizer for language models. *arXiv preprint arXiv:2407.07972*, 2024.

A. Model Architecture

A.1. Learning Rate Tunes and Extrapolation

To scale both our transformers and LSTM, we run a learning rate tune at least 4 model sizes over the range: $[10^{-3}, 10^{-2.5}, 10^{-2}, 10^{-1.5}, 10^{-1}]$. We then do a more fine-grained learning rate tune in intervals of $10^{0.25}$, ensuring that equal hyperparameter tuning resources are used for all our models (Modern Transformer, Retro Transformer, LSTM). Afterward, we fit a power-law trend between the learning rate and model hidden dimension and use that trend to determine the optimal learning rate at all sizes.

A.2. Further Discussion of Hyperparameter Choices Including Dropout

We choose to use no dropout for both our LSTM and Transformer as both perform considerably better without dropout. To check this for LSTM we compared an LSTM with hidden dimension 64 with the dropout settings recommended by Melis et al. (2017) to an LSTM with no dropout and tune learning rates for both. The best performing LSTM with dropout has a validation loss of 6.5, while the best performing LSTM without dropout has a validation loss of 5.9.

A.3. Data Selection, Loading, and Size

We choose to use a subsample of the C4 dataset for both the transformer and LSTM. The size of dataset we use for training is based on the model size and our token-to-parameter ratio. In order to better outline the optimal compute frontier, we choose to use a token-to-parameter ratio of 40 in our scaling experiments. This is to ensure that we do not stop early before compute-optimal level of data. Stopping early would change the frontier envelope, while stopping late would not change the frontier, as overtrained models would pass the frontier. While for our ablation experiments, we use a token to parameter ratio of 20 as recommended by Hoffmann et al. (2022).

Table 4. Base hyperparameters used to train the LSTM model, mostly taken from Melis et al. (2017). This includes TBPTT length, weight decay, number of layers, batch size, etc. Melis et al. (2017) recommends setting β_1 to 0, we find dropout is not helpful, see Appendix A.2

Hyperparameter	Value
Number of layers	1
TBPTT length	32
TBPTT stride	32
Sequence length	128
Batch size	64
Input dropout	0.0
Hidden dropout	0.0
Output dropout	0.0
Weight decay	10^{-4}
Gradient clipping	1.0
Learning rate schedule	cosine annealing
Min learning rate	$0.1\times$ max lr
Warmup fraction	10%
Default optimizer	AdamW
β_1	0.0
β_2	0.999
Initialization	LSTM (orthogonal hidden matrices)
Embedding weight tying	True
Tokenizer	GPT-2 Tokenizer
Vocabulary	50257
Dataset	C4
Validation set size	500k tokens
Token-to-parameter ratio	40

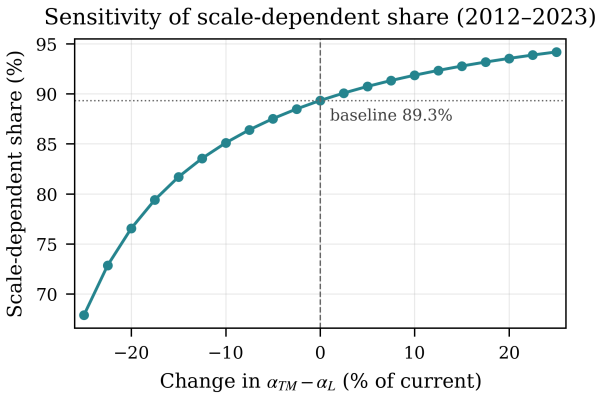

Figure 5. Graph depicts how changing the difference between Transformer α_{TM} and LSTM α_L scaling exponents leads to slightly-different levels of scale-dependent attribution.

B. Sensitivity Analysis

A key parameter in our model of algorithmic progress is the scaling-exponent difference between Transformers (α_{TM}) and LSTMs (α_L). We vary this difference and measure the share of algorithmic gains attributable to the scale-dependent LSTM-to-Transformer transition. Our results are robust to variations in either direction: a 25% increase in the exponent difference shifts the attributed share from 90% to 95%. This analysis is shown in Fig 5

B.1. Sensitivity to Irreducible Loss

We use a fixed Irreducible loss of 1.9 for our scaling law fits, this is fit and analyzed in Appendix G. However, within a reasonable range, our choice of irreducible loss has relatively little effect on the scaling exponents we determine.

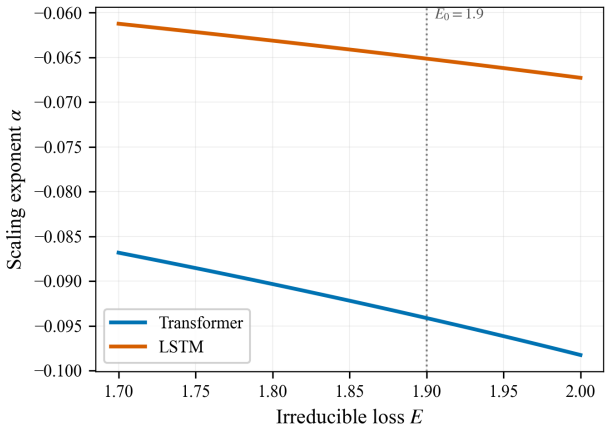

Figure 6. Sensitivity of measured scaling exponents for LSTM and modern-transformer to choice of irreducible loss.

C. Scaling-Experiment Models and Hyperparameters

This appendix records the model definitions and hyperparameters used in Section 4.1.1. Unless otherwise stated, all models are trained on the same dataset and tokenizer, with matched optimization and tuning budgets across model families.

C.1. Model Definitions

Modern Transformer. Decoder-only Transformer with contemporary default components: rotary positional encodings, RMSNorm prenorm, and SwiGLU. We keep a constant width-to-depth aspect ratio across scales (chosen for stability in the small-model regime), and set the attention-head configuration to maintain a small head dimension appropriate for these sizes. Weight initialization follows the standard GPT-2/BERT-style recipe (zero biases; Gaussian weights). FLOPs are measured using a PyTorch profiler on one minibatch and scaled by the number of optimizer steps and gradient-accumulation schedule.

Retro Transformer. Matches the Modern Transformer architecture and scaling rules but reverts four components to approximate an early Transformer-style recipe: (i) SwiGLU $\rightarrow$ GeLU, (ii) RMSNorm prenorm $\rightarrow$ LayerNorm postnorm, (iii) rotary $\rightarrow$ sinusoidal positional encodings, (iv) cosine decay $\rightarrow$ inverse square-root decay.

Kaplan Transformer (analytical scaling policy). The Kaplan Transformer is *not* a separate architecture. It is the Retro Transformer evaluated under Kaplan-style parameter–data allocation via an analytical adjustment described in Appendix E. We use this to isolate the effect of scaling policy from architectural changes.

LSTM baseline. Vanilla LSTM based on the strong pre-Transformer recipe of Melis et al. (2017). We use one LSTM layer throughout scaling for the reasons discussed in Section 4.1.1. We use Xavier-uniform embeddings, orthogonal recurrent matrices, and a forget-gate bias of $+1$. Following Melis et al. (2017); Merity et al. (2017), the original recipe uses variational dropout; in our internet-scale regime we set dropout to zero (as in common Transformer practice (Hoffmann et al., 2022)) and tune learning rates comparably across families.

C.2. Scale Grid and Parameter Ranges

We scale models by varying hidden size on a fixed grid and holding aspect ratios constant within each family. In Section 4 we use hidden dimensions

$$[32, 64, 96, 128, 160, 192, 224, 256],$$

corresponding to Transformer sizes of approximately 1.6M–29M parameters and LSTM sizes of approximately 1.6M–11.2M parameters (exact counts depend on embedding and vocabulary parameters).

C.3. Optimization, Data, and Tuning Protocol

All models use the same tokenizer (GPT2 Tokenizer) and dataset (C4) and the same validation protocol. We run matched learning-rate tuning across model families: we evaluate a coarse learning-rate grid at multiple sizes, then fit a smooth learning-rate vs. hidden-size relationship used to set learning rates at intermediate sizes. Full tuning details are in Appendix A.1. Where relevant, we report compute using measured FLOPs rather than step counts.

D. Reproducibility Statement

All code used in model training and model analysis, along with setup instructions, is hosted here: `https://anonymous.4open.science/r/Experimental_Progress-70B8/README.md`. The main body of our text contains details on our training and analysis procedure. A detailed list of hyperparameters used is in Appendix A.

E. Kaplan-Style Scaling When the True Frontier is Chinchilla-Optimal

In Figure 7 below, we show loss curves from using Kaplan and Chinchilla's recommendations for compute optimal scaling (specifically optimal scaling by analytically solving the full parameter model). We take the Chinchilla functional form for model loss $L(N, D)$ as ground-truth, only modifying the allocation of parameters and dataset size between each curve. Unsurprisingly, Chinchilla scaling is a perfect power law.

We use the constants $E = 1.69$, $A = 406.4$, $B = 410.7$, $\alpha = .34$, and $\beta = .28$ and include the functional form below for the reader's convenience.

$$L_{\text{Chinchilla}}(N, D) = \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + E \tag{1}$$

$$= \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.28}} + 1.69 \tag{2}$$

For the compute allocation, Chinchilla scaling explicitly defines optimal scaling of N and D to be

$$N_{\text{Chinchilla}}(C) = G\left(\frac{C}{6}\right)^{\beta/(\alpha+\beta)} \tag{3}$$

$$D_{\text{Chinchilla}}(C) = \left(\frac{1}{G}\right)\left(\frac{C}{6}\right)^{\alpha/(\alpha+\beta)} \tag{4}$$

where $G = (\frac{\alpha A}{\beta B})^{1/(\alpha+\beta)}$. By construction, this gives a precise power law with a well defined inverse:

$$\text{ChinchillaCompute}(L) = \frac{5.4 \times 10^{19}}{(L - 1.69)^{6.5}} \tag{5}$$

Finally, we infer the full Kaplan scaling recommendation: Figure 14 in Kaplan et al. (2020) gives a functional form

for compute optimal dataset scaling, and after applying unit conversions and using the relationship $C = 6ND$, we get the following:

$$N_{\text{Kaplan}}(C) = (3.6 \times 10^{-6})C^{0.73} \tag{6}$$
$$D_{\text{Kaplan}}(C) = (4.6 \times 10^{4})C^{0.27} \tag{7}$$

Together, these give us a closed form for the compute multiplier when rebalancing from Kaplan to Chinchilla scaling for a fixed C. We call this function M, and compute it as follows:

$$\begin{aligned} M(C) &= \frac{C}{\text{ChinchillaCompute}(L_{\text{Chinchilla}}(N, D))} \\ &\quad \text{where } N = N_{\text{Kaplan}}(C),\ D = D_{\text{Kaplan}}(C) \\ &= (1.85 \times 10^{-20})(C)\left(\frac{28737.9}{C^{0.2482}} + \frac{20.337}{C^{0.0756}}\right)^{6.512} \end{aligned} \tag{8}$$

As C gets large (roughly 10^{23} FLOPs), we can use a power law to approximate this expression:

$$M(C) = (6.13 \times 10^{-12})(C^{0.508})$$

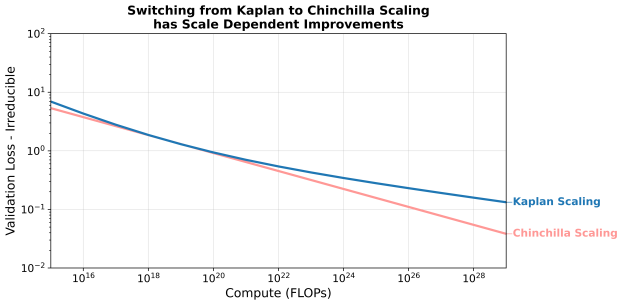

Figure 7. Analytical difference in performance between transformer models scaled with Kaplan versus Chinchilla recommendations. Interestingly, the efficiency gap first converges, then diverges.

F. Transformer SGD Scaling Study

In order to train a Transformer with SGD we use a momentum of .98 and 0 weight decay as recommended by Srećković et al. (2025). We use the same scaling procedure as our default Transformer, and all other hyperparameters are held constant. We find little scaling difference between Transformers trained with SGD and Transformers trained with AdamW. However, in our experiments, transformers trained with SGD are notably more unstable. For instance, our SGD training curve has a slight concavity early on.

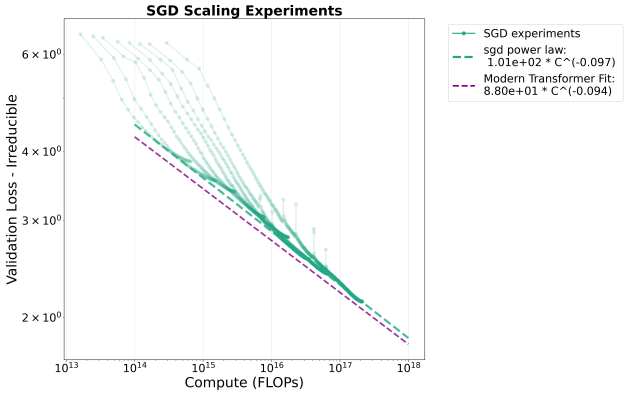

Figure 8. Optimizers have little effect on scaling exponents. The blue dashed line represents the compute-optimal scaling fit for our Modern Transformer trained with SGD. The purple dashed line represents the experimental fit for our Transformer trained with AdamW.

G. Irreducible Loss

The irreducible loss represents the self-entropy of the data distribution. This means it varies from dataset to dataset but should be constant across models. In our paper, we set the irreducible loss to 1.9. In contrast, Hoffmann et al. (2022) fits an irreducible loss of 1.69 using MassiveText (similar to C4) Besiroglu et al. (2024) finds an irreducible loss of 1.8 using Hoffmann et al. (2022) data. Porian et al. (2024) did a replication of Kaplan and Chinchilla scaling on RefinedWeb and OpenWebText2. They find irreducible loss fits from 1.68 to 2.01. Muennighoff et al. (2023) finds an irreducible loss of 1.87 for Transformers trained on C4. In light of this variation, we use scikit learn to fit the optimal frontier to the form $L = E + AC^{-\alpha}$, restricting E to lie between 1.3 and 2.2. For our Modern Transformer, 2017 transformer, and LSTM, this approach yields an irreducible loss close to 1.9 for all the models used in our study.

H. Formal Definition of CEG Functions and Multipliers

In order to more precisely discuss algorithmic progress, we introduce a generalized notion of the standard compute equivalent gains (CEG) framework (Davidson et al., 2023; Ho et al., 2024; Sanderson et al., 2025). Our approach varies from much of the existing literature by allowing algorithmic innovations to have different efficiency gains across compute scales. More precisely, we define the **CEG function** and **CEG multiplier** distinctly:

Definition (CEG function). For two algorithms A, A' and a fixed performance metric (e.g. loss), the **compute equivalent gain function** from A to A' is a function f such that a model trained with A using compute C and a model trained with A' using compute $C/f(C)$ reach equivalent

performance, for all C in the domain of f.

Fixing the compute budget for one of the algorithms, we can recover the traditional definition of a CEG multiplier:

Definition (CEG multiplier). Let M be a model trained with algorithm A using compute C. Let f be the CEG function between A and some A'. Then $f(C)$ is the **CEG multiplier** of A' *relative to* M.

Intuitively, CEG functions compare training algorithms across compute scales, whereas CEG multipliers compare models by fixing a reference compute (or performance) level.[3] This distinction is central to our analysis: when algorithms exhibit scale dependence, reporting only CEG multipliers can induce misleading trends.[4]

Edge cases. If an algorithm plateaus in performance, the CEG function may be undefined at some thresholds (informally requiring "infinite" compute) (Erdil & Besiroglu, 2022). Finally, if the CEG function is known to be constant between two algorithms, one may choose any common performance threshold to estimate that constant value. For our ablation experiments, we argue the ablated innovations admit approximately constant CEG functions within our compute regime, and we use the lowest negative log-likelihood threshold achieved by all ablations (NLL$= 5.3$) when reporting these multipliers.

H.1. Interaction Between Algorithms

Conventional theory of algorithmic gains presupposes that most algorithmic gains are multiplicative (Ho et al., 2024; Davidson et al., 2023). For instance, if algorithm A increases training efficiency by $2\times$ and algorithm B increases training efficiency by $3\times$, respectively, then the combination of algorithm A and B should have a training efficiency gain of $6\times$. However, across architectures this is not the case. For example, Adam provides a computational efficiency gain for transformers, yet most state-of-the-art LSTMs relied on RMSProp or plain SGD over Adam (Merity et al., 2017; Melis et al., 2017; Keskar & Socher, 2017).

To help identify these interactions, we train a transformer model similar to the original architecture developed by Vaswani et al. (2017). In this "Retro" Transformer, four algorithmic improvements are reverted. We switch from SwiGLU to GeLU, pre-RMSNorm to post-layernorm, rotary encoding to sinusoidal encodings, and cosine decay learning rate schedule to square root decay schedule. Using the estimates in Figure 1 and assuming multiplicative

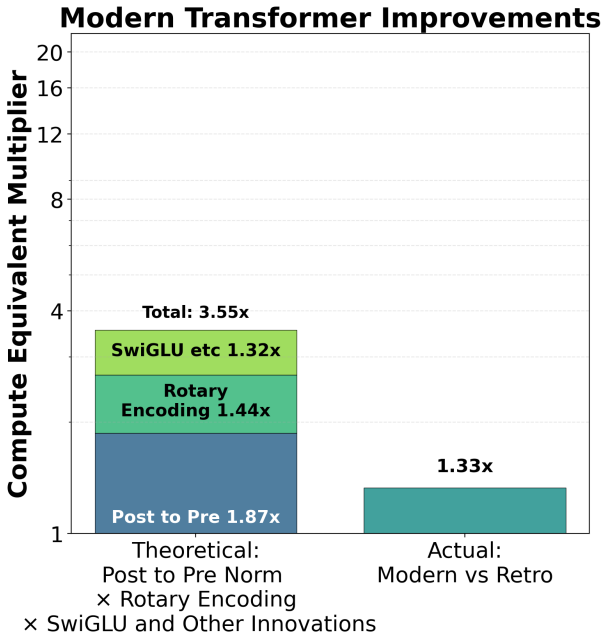

Figure 9. Comparison of the total effect of all measured post-Transformer algorithmic changes (left bar) vs. multiplying ablation-estimated effects together.

improvement we would expect CEG gains of $3.43\times$. However, the efficiency gain is only $1.33\times$. In addition, we compare our default or "Modern" transformer (defined in Appendix C.1) to our LSTM baseline. At a loss threshold of 5.3, we find a combined efficiency gain of $6.28\times$, whereas multiplying the individual improvements together, along with the LSTM to Retro Transformer improvements, yields a substantially higher predicted gain of $16.66\times$.[5]

This suggests that multiplying individual improvements, estimated from ablation experiments, overestimates the impact of algorithmic improvements. We hypothesize the following explanation by analogy: consider replacing the engine of a modern car with a *Model T* engine. A *Model T* engine in an otherwise modern car might perform far worse than it did in the original *Model T* due to system incompatibilities (similarly with any other component). As a result, one would overestimate the proportion of progress for each of these inventions independently. Similarly, it may be that replacing algorithmic components individually appears to cause greater efficiency loss than replacing all components at once. In this sense, algorithmic improvements are sub-multiplicative and have slightly non-orthogonal effects.

[3] "Training algorithms" includes everything except compute, e.g. architecture, optimizers, and data–parameter balancing.

[4] In our ablation experiments, we report CEG multipliers between algorithms using compute-optimal performance within our hyperparameter search.

[5] In the case where we have scaling fits, we use the CEG between scaling fits to be consistent with Figure 2. If we use individual training curves, we would get a Modern-to-Retro Transformer difference of $2.7\times$ and a Modern-to-LSTM difference of $5.9\times$

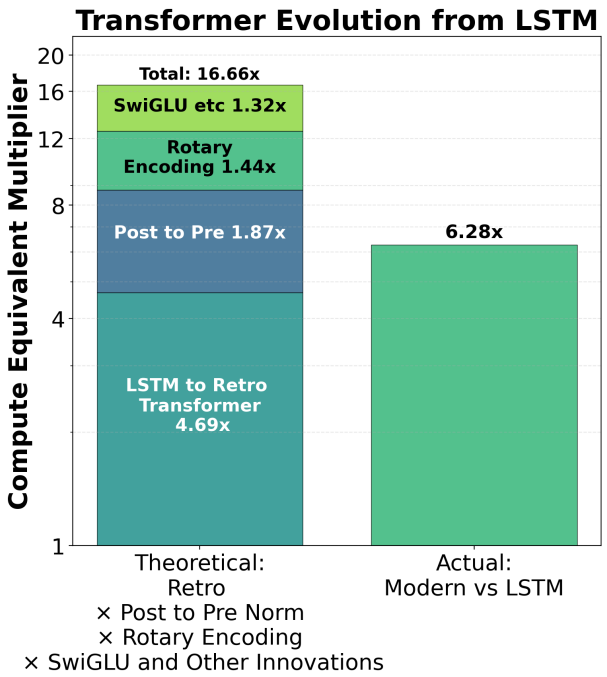

Figure 10. Comparison of the total effect of all measured changes (including the Transformer transition) vs. multiplying ablation-estimated effects together.

I. Ablation Experiment Details and Component-Level Results

For our experimental approach, we examine a small 3.6M parameter Transformer with rotary-based encodings, a constant width-to-depth ratio, and a GeLU activation function. Our baseline is identical to our "modern" Transformer, described in our scaling experiments (Appendix C.1). For each ablation, we run learning-rate tunes by testing the default learning rate ($10^{-2.25}$) and $[10^{-2.5}, 10^{-2}]$.[6] As mentioned previously, we choose the lowest loss threshold that all ablated models are capable of (in our case, a loss of 5.3) to determine the CEG multiplier. All of our ablation experiments use a token-to-parameter ratio of 20 as in Hoffmann et al. (2022).

Activation Functions. We test the impact of new activation functions like SwiGLU over the traditional GeLU activation function. In line with Shazeer (2020), we find that SwiGLU has a measurable $1.17\times$ efficiency gain over GeLU even taking into account its increased parameter requirements.

Positional Encoding. We test three different types of positional encoding: rotary, sinusoidal, and learned positional encodings. We estimate that rotary encoding constitutes an improvement of 44% in training efficiency over sinusoidal encoding originally employed in Vaswani et al. (2017). This is in contrast to Sanderson et al. (2025), which estimates a 70% gain, and consistent with Biderman et al. (2021), which sees a 10%–30% training efficiency gain. We expect this effect to be sequence-length dependent.

Learning-Rate Schedules. We test three prominent learning-rate schedules: linear warmup/inverse square-root decay (Raffel et al., 2020; Vaswani et al., 2017), linear warmup/linear decay (Devlin et al., 2019), and linear warmup with cosine decay (Brown et al., 2020). We observe very small training-efficiency differences between schedules ($< 5\%$), consistent with Kaplan et al. (2020).

Normalization Techniques. We test a pre-RMSNorm model, a pre-layernorm model, and a post-RMSNorm model. Pre-layernorm(RMSNorm) applies layer normalization before attention in the residual stream and is generally seen as more stable than post-layernorm (Xiong et al., 2020). RMSNorm removes mean-centering and can improve runtime (Zhang & Sennrich, 2019). We observe substantial improvements of 87% transitioning from post-RMSNorm to pre-RMSNorm, with more modest improvements of 9% switching from pre-layernorm to pre-RMSNorm.

Optimizers. Prior work has shown that SGD performs notably worse on Transformers compared to Adam (Ahn et al., 2023; Zhao et al., 2024), potentially due to Hessian heterogeneity (Zhang et al., 2024). More recent results suggest the gap can narrow under specific hyperparameters (Srećković et al., 2025). With batch size 64, momentum 0.98, and no weight decay (as recommended by Srećković et al. (2025)), we observe a $1.87\times$ gap in training efficiency compared to AdamW. We further examine optimizer scaling in Appendix F. We compare Adam to AdamW and find a negligible difference in our setting.

J. Literature Estimates for Hard-to-Ablate Components

Why Include Literature Estimates? In addition to ablation experiments, we include conservative bounds from the literature for mixture-of-experts (MoE) training and tokenization methods. MoE models are challenging to implement and measure at small scales due to routing and loss-balancing, and tokenizers are difficult to assess using perplexity because perplexity is inherently tokenizer-dependent.

Mixture of Experts. MoE is often discussed as an inference improvement, but it can also improve training efficiency. We approximate MoE training gains as $\leq 2\times$

[6] If the optimal learning rate differs from the default, we test learning rates outside this range by factors of $\sqrt{10}$.

based on existing comparisons: Riquelme et al. (2021) report slightly less than $2\times$; isoflop comparisons in He (2024) suggest substantially less than $2\times$; Li et al. (2025) report approaching dense performance with roughly twice the compute; Muennighoff et al. (2024) report larger gains but without controlled isoflop comparisons; and Tay et al. (2023) find similar scaling behavior with noisy results. Given uncertainty, we leave a dedicated MoE scaling study to future work.

Tokenizers. Early language models used word- or character-level tokenization. Subword tokenization (BPE, WordPiece, SentencePiece) mitigated vocabulary/sequence-length tradeoffs. Ali et al. (2024) finds inefficient tokenization can increase training cost by 68%. We do not include tokenizer gains in our main aggregations due to uncertainty and limited standardized historical comparisons.

K. Additional Discussion: Skewed Distribution and Interaction Effects

Experimentally, we find total measured efficiency gains switching from an LSTM to a Modern Transformer of $6.28\times$, while switching from an LSTM to our Retro Transformer yields $4.69\times$. This is smaller than estimates from Ho et al. (2024), which report $\sim 60\times$ for the LSTM-to-Transformer transition at their reference points. Although we see moderate $\sim 2\times$ gains for some changes (e.g., optimizer and normalization), most measured improvements are small, and the distribution of multipliers is uneven and concentrated in a handful of changes. This motivates our scaling experiments in Section 4, where we test whether the dominant historical gains arise from scale-dependent innovations.