Efficient Pre-Training with Token Superposition

Bowen Peng*
Nous Research
bloc@nousresearch.com

Théo Gigant*
Nous Research
theo@nousresearch.com

Jeffrey Quesnelle
Nous Research
emozilla@nousresearch.com

Abstract

Pre-training of Large Language Models is often prohibitively expensive and inefficient at scale, requiring complex and invasive modifications in order to achieve high data throughput. In this work, we present Token-Superposition Training (TST), a simple drop-in method that significantly improves the data throughput per FLOPs during pre-training without modifying the parallelism, optimizer, tokenizer, data, or model architecture. TST is done in two phases: (i) A highly efficient superposition phase where we combine many contiguous tokens into one bag and train using a multi-hot cross-entropy (MCE) objective, and (ii) a recovery phase where we revert back to standard training. We extensively evaluate TST on the scale of 270M and 600M parameters and validate on 3B and a 10B A1B mixture of experts model, demonstrating that it is highly robust in different settings. Ultimately, TST consistently outperforms baseline loss and downstream evaluations, and under equal-loss settings, TST yields up to a 2.5x reduction in total pre-training time at the 10B A1B scale.

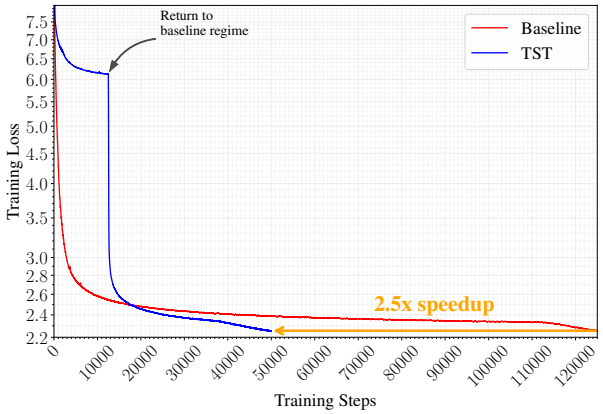


Figure 1: Loss curves during the pre-training of two Qwen3-like MoE models (10B-A1B) with baseline pretraining and token superposition training (TST) where we stop the training early to match an equal training loss. The baseline training sees 1.05T tokens while the TST training sees 2T data tokens. Every step in all conditions are equal-FLOPs, thus the speedup can be directly computed w.r.t. the number of steps. More details are in Table 1.

*Equal contribution.

1 Introduction

The rapid proliferation of modern generalist Large Language Models (LLMs) has been driven not only by increases in model size, but critically, by aggressive data scaling [53, 21, 6, 34, 5, 57, 1], with recent training regimes often overtraining [16] well beyond compute-optimal estimates to maximize performance at inference time. In this data-hungry paradigm, one of the major concerns during pre-training is the efficiency in which raw text is consumed given a fixed amount of compute.

Recent advances in language modeling pre-training efficiency can be broadly organized into three categories:

1. **Information maximization**: maximizing the information given per sample through improved input priors and representations (better tokenization, BPE [48], SuperBPE [35], Unigram [29], n-gram hashing [36, 9]) and richer training signals (auxiliary losses, including multi-token prediction [20, 34], order-augmented objectives [64]).
2. **Compute sparsity**: keeping the input representation fixed but reducing the FLOPs required to process each token by activating only a subset of parameters or attending to a subset of positions. This constitutes a *compute-level* prior: similar expressivity, less work per token (sparse mixture-of-experts [26, 13], sparse attention [59, 58]).
3. **Compressive modeling**: learning to *further compress* the representation within the model itself, reducing the number of representations that flow through the expensive layers. This constitutes a *representation-level* prior: fewer tokens internally, but dense computation on each (e.g., Bolmo [44], H-Net [25], Byte-Latent Transformer [46], Autoregressive UNet [54] Perceiver-Resampler [2]).

Note that some of these works also touch on inference-time efficiency, which we acknowledge as an important line of work but orthogonal to our focus on pre-training efficiency. While some methods above confound and mix both training-time and inference-time efficiency, other methods focus on inference-time efficiency only (*e.g.* diffusion [45], speculative decoding [31]), and almost none of them have decoupled the training-time efficiency and focus solely on this latter part (except concurrent work from Zheng et al. [62]).

Many recent works refute the training-time and inference-time efficiency equivalence, where they show that scaling up inference-time compute independently of training time compute can improve performance on downstream tasks (e.g. CoT/reasoning models [55], ParScale [8], looped language models [63]), thus it is important for our method to be only used during training, and keep the model architecture and expressivity "untouched" for inference compared to the baseline in order to minimize any confounding factors.

Furthermore, recent evidence suggests that the performance advantage of subword-based over byte-level language models is largely driven by increased training sample throughput [18]. Building on this insight, we investigate whether LLM training efficiency can be further optimized by maximizing training-time throughput independently of the model's inference-time architecture.

Finally, the monolithic pre-training paradigm is currently shifting towards two-stage or multi-stage pre-training schemes. Many recent works have found that better or more efficient training methods can be used for the first stage of training [44, 23, 30], and then find that the model is elastic enough to quickly adapt to the final desired model behavior, often saving on training cost and/or resulting in a better quality model. However, most prior works focus on the mid-training or post-training regime, but we contextualize this work by showing that the same ideas can be applied to pre-training.

We introduce Token-Superposition Training (TST), a method that tries to improve training efficiency by increasing token throughput while still priming the model for the desired task of autoregressive prediction. Our work starts with this fundamentally different perspective:

Perspective

Can we improve pre-training efficiency by forcing a higher token throughput during training, without modifying the final model architecture and its inference dynamics?

After completion of this work, we became aware of Shao et al. [50], which independently proposed the same core mechanism: averaging consecutive token embeddings as input, predicting all tokens in the next group via cross-entropy with a single head, and transferring to standard token-level training in a second phase, with only minor differences from ours on the algorithm side. Although our work converges to a very similar method, we propose a different view on the problem and extend the research in several directions, with more details in Section 2.4.

2 Related works

2.1 Alternative Prediction Objectives

In contrast to standard autoregressive next-token prediction, several studies have explored alternative training objectives to improve representation learning. Tay et al. [52] introduced the *mixture-of-denoisers* framework, which unifies diverse denoising tasks—such as span corruption and causal language modeling to provide superior generalization across architectures. In the context of encoder models, Gisserot-Boukhlef et al. [19] demonstrated that a two-stage pretraining schedule, transitioning from causal language modeling to masked language modeling (MLM), outperforms standard MLM baselines in some downstream tasks.

2.2 Auxiliary Prediction of Future Representations

Recent pre-training literature introduced auxiliary losses to move beyond single-token targets, with the goal of increasing the information density per gradient step. Gloeckle et al. [20] introduced *multi-token prediction* (MTP), using k independent heads to predict the next k tokens simultaneously. Although MTP improves sample efficiency in some cases and has been featured in major state-of-the-art LLM pre-training runs [34], it has limited benefit to smaller models and requires the tuning of an extra hyper-parameter k while introducing additional parameters.

Other approaches explore using auxiliary losses with targets using representations involving future tokens. DeepseekV3 [34] uses a modified version of MTP, featuring cascaded predictions using additional MTP modules. Zuhri et al. [64] replace MTP with the prediction of the relative order of future tokens, relaxing the complexity of MTP by simplifying the task and requiring only a single additional head. Liu et al. [37] proposed *next concept prediction* with predicted concepts covering segments spanning multiple tokens. Mahajan et al. [41] introduced *future summary prediction*, where the model predicts a compressed representation of future tokens utilizing either hand-crafted bag-of-words representations or learned latent features from a reversed sequence model. Notably, an entry within the `modded-nanogpt` speedrun [27] has implemented an MTP-inspired loss that shares conceptual similarities with the *next bag-of-tokens* prediction explored in our work. These works illustrate that additional signal involving predicting representations of future tokens can yield substantial gains in pre-training sample efficiency.

2.3 Input Granularity

Input granularity is a fundamental hyperparameter in both vision and language modeling. In Vision Transformers [14], patch size serves as a primary control for the trade-off between FLOPs and performance. Anagnostidis et al. [4] showed that scheduling patch size from coarse to fine-grained during training improves isoFLOPs performance for Vision Transformers. In LLMs, this granularity is typically set by the tokenizer's *fertility*, *i.e.* the average number of tokens required to represent a word. Subword tokenizers provide a coarser-grained view of text compared to byte-level or unicode representations. Liu et al. [35] further merge BPE tokens into supertokens, resulting in coarser tokens exhibiting improved training performance compared to regular BPE. Recent work by Minixhofer et al. [44] explored coarse-to-fine distillation of LLMs by resuming subword-level pretraining with byte-level objectives. Furthermore, Zheng et al. [62] demonstrated that including mixed granularities at training-time, through both compressed and raw byte representations, resulted in better byte-level performance compared to raw byte representation only. Gigant et al. [18] investigated the performance gap between subword and byte-level models, attributing the subword advantage largely to increased *sample throughput* at isoFLOPs, a direct consequence of the coarser tokens resulting from the sequence compression effect of subword tokenization. These findings suggest that coarser training-time input granularity is a key driver of modern LLM training speed. In this work, we lever-

age both the additional signal from future tokens representations and a coarser input granularity at training-time to improve the model pre-training efficiency. Crucially, during the second phase of training, we return to the baseline granularity and loss.

2.4 Patch-Level Training

Most directly related to our work, Shao et al. [50] introduced patch-level training for LLMs, in which consecutive token embeddings are averaged into "patches" and the model trained to predict all tokens in the next patch using a single output head with cross-entropy loss. After patch-level training, the model reverts to standard token-level training. This is algorithmically identical to what we term Token Superposition Training, but with major differences in background theory and execution, which we attribute to conceptual convergence. Shao et al. [50] frame patch-level training as reducing total FLOPs for a fixed dataset, whereas we propose TST as two orthogonal but additive processes that increase token throughput at constant per-step FLOPs (Section 5.2) following the throughput hypothesis of Gigant et al. [18], where we make careful implementation decisions that result in accurate comparisons which simplifies scaling, and justify the choice of the multi-hot loss function based on empirical evidence (Section 3.2). Shao et al. [50] demonstrated the approach with trainings up to 2.7B parameters on 360B tokens, achieving a total training cost of $0.5\times$. We extend this with an extensive search of the hyperparameter space and demonstrate successful scaling at the 10B parameters, 2T token scale. They also observed that maintaining the same architecture (without additional projection layers) across both phases is critical for successful recovery. This finding aligns with our representation alignment analysis (Section 5.3).

3 Methodology

Token Superposition involves two small changes when compared to baseline next-token prediction training, illustrated in Figure 2. In the Token Superposition Training (TST) regime, an LLM processes superpositions of token embeddings obtained via averaging the embeddings of contiguous tokens in non-overlapping s-grams. Each such superposed latent token, which we call "s-tokens", results in a single prediction, compared against the next non-overlapping bag-of-tokens.

The superposition objective is semi-causal and semi-autoregressive. In other words, the model still broadly predicts the input sequence from left to right, but we lose the ordering of the tokens in the superposition bag and the sampling capabilities during inference. Not surprisingly, if used as-is, a model trained using only TST has nonsensical outputs which represent a mixed probability of any of the s future tokens. In order to remedy this, we opt for the simplest method which is to revert to training with the standard next-token causal prediction objective after some amounts of steps, where we define r as the ratio of steps where we train with TST. In the second stage, we resume training from the saved checkpoint with the TST code fully removed to avoid any possibility of experimental and results contamination.

We leave more complex conversion methods or other potential uses cases for TST as future work, where for example a model trained with TST might have informative latents that can be used to efficiently predict or verify multiple tokens at once, or using a TST-ed model as a compressive prior or encoder model.

3.1 Input Superposition: Bag-of-Token-Embeddings

First, the contiguous tokenized data sequence of shape $B\times L\times V$ is segmented into **non-overlapping** contiguous segments of s tokens, called bags. Here, B denotes the batch size, L the data token sequence length and V the vocabulary size of the tokenizer. This gives us a bagged view of the data in the shape of $B\times l\times s\times V$, where s is the superposition bag size and l the latent "s-token"sequence length.

In the embedding layer of the model, a single latent "s-token" is created by superposition of the tokens in a bag via the average of their embeddings, resulting in a final shape of $B\times l\times d$, where d is the residual dimension of the model. The code is shown in Appendix A.

As the model performs computations over a coarser-grained representation of the input text, it processes s times more data tokens for every FLOPs used on the latent "s-tokens". Therefore, in order

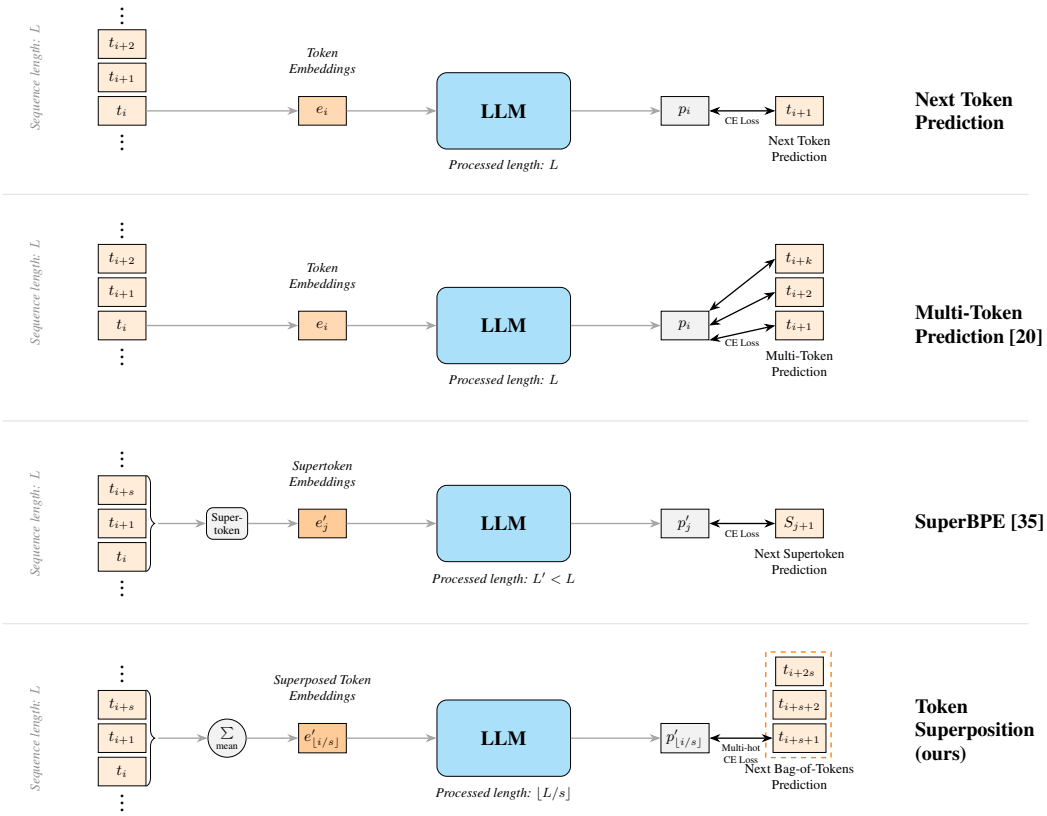


Figure 2: Comparison between standard next token prediction, TST and a few methods that superficially resemble TST. Note that this comparison is illustrative only for the purpose of understanding TST.

to make a valid comparison during training, we choose to make every TST step equal-FLOPs to the baseline training by increasing the data sequence length L by s times during the superposition phase.

This is what allows the model to ingest tokens at a higher rate during the superposition phase for the same amount of FLOPs per step compared to standard training. A faster alternative would be to increase the micro batch size by s instead, but it is not explored in this work.

3.2 Output Superposition: Next Bag-of-Tokens Prediction with Multi-hot Cross-Entropy Loss

Second, to predict the next "bag of tokens" instead of a single token with a single output head, we modify the standard one-hot cross-entropy (CE) loss into a **multi-hot** cross entropy (MCE) loss.

We have the standard CE loss with respect to the predicted logits $\mathbf{z}$ and the label index y:

$$\mathcal{L}_{\text{CE}}(\mathbf{z}, y) = -z_y + \log \sum_{i=1}^{V} \exp(z_i) \tag{1}$$

One possible expansion of this one-hot CE loss into a multi-hot MCE loss is simply to target an equal probability $1/s$ for each valid label (that together sum to 1). In our case, the label bag size $|\mathbf{y}|$ is equal to s. With some rearranging, we get the following:

$$\mathcal{L}_{\text{MCE}}(\mathbf{z}, \mathbf{y}) = \frac{1}{|\mathbf{y}|} \sum_{y \in \mathbf{y}} \left(-z_y + \log \sum_{i=1}^{V} \exp(z_i) \right) - \log |\mathbf{y}| \tag{2}$$

The expanded derivation is shown in Eq. 4, Appendix C.

If we do not care about the absolute loss value during training and only care about the training dynamics, we can drop the $\log |\mathbf{y}|$ term from the loss, as it's gradient is 0. This yields the final simplified form that we use:

$$\mathcal{L}_{\text{MCE}}(\mathbf{z}, \mathbf{y}) = \frac{1}{|\mathbf{y}|} \sum_{y \in \mathbf{y}} \mathcal{L}_{\text{CE}}(\mathbf{z}, y) \tag{3}$$

Finally, the standard next token prediction labels are shifted to the left by $s-1$ before splitting into non-overlapping bags to preserve causality. This ensures that each bag of s tokens at the positions $[t, t+s-1]$ predicts the next bag of tokens at $[t+s, t+2s-1]$.

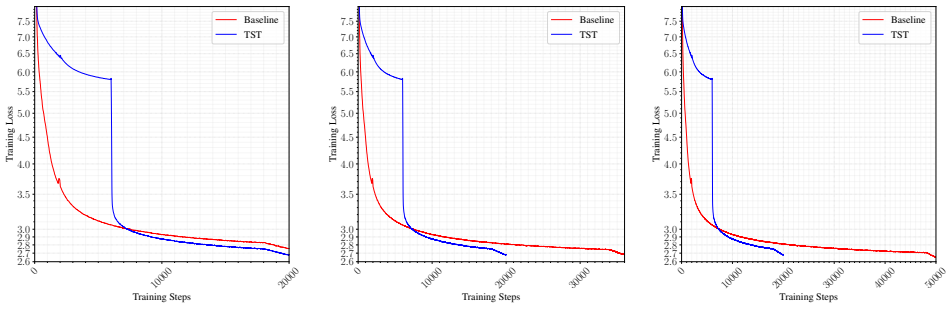
The simplified form allows us to take advantage of the highly optimized and compiled CE loss kernels that exist in major pre-training libraries and use them without major modifications to the training code. Further optimization can be done to reuse the inner $\log \sum_i \exp(z_i)$ term, but that is not explored in this work as the overhead to summing outside the loss with a for loop is negligible and uses no additional memory. The code is shown in Appendix A.

We also tried many variants of possible multi-hot bag losses, such as Hinge loss [12] and Binary Cross-Entropy (BCE) loss [49]. But these results were significantly worse than the MCE loss we picked and even worse than the baseline training without TST. Although an interesting alternative to the MCE loss shown here was explored, with more details in Appendix C.2, the final chosen MCE loss has the best balance between soundness and simplicity.

4 Experiments

We perform a battery of experiments that cover a wide range of superposition settings, varying the superposition bag size and ratio at different model scales and total durations. For all trainings, the TorchTitan [33] pre-training library was used with FSDP [61] parallelism, running on 64 NVIDIA B200 GPUs for the bigger models, and 8 B200 GPUs for the smaller models.

For pre-training the smaller models, we broadly follow the training procedures as outlined in SmolLM [3]. For the 270M and 600M parameter models, we adapt and use the shape of the SmolLM2 models to fit the Llama3 [21] modeling code, with two modifications: we use the Llama3-8B tokenizer, and we do not tie the weights of the input embeddings to the output head. All other settings were kept the same (including layers, LM heads, inner dimensions, etc.). The untied embeddings make our models bigger, with the 270M matching SmolLM2-135M and the 600M matching SmolLM2-360M. Similarly for the 3B run, we matched it with SmolLM3-3B with the same modifications applied. Finally, the DCLM [32] dataset was used as the pre-training dataset, with a standard batch size of 2M tokens[1] for the SmolLM-like models. More detail is shown in Table 1.



(a) Equal-FLOPs (b) Equal-Loss (c) Equal-Data

Figure 3: Same constraints comparisons between different baseline training and one Token Superposition Training, using superposition bag size $s = 6$ and step ratio $r = 0.3$. These are the four 3B parameter runs described in Table 1.

[1] or "s-tokens"in the case of TST, in which we call both "equivalent-tokens" in the figures.

Model	Params	TST Steps	Total Steps	TST Bag Size	TST Tokens	Total Tokens	B200-Hours (↓)	Final Loss (↓)	HellaSwag (↑)	ARC-E (↑)	ARC-C (↑)	MMLU (↑)
Dense Baseline	270M	–	20000	–	–	42B	34	3.212	36.3	46.7	24.9	–
Dense TST	270M	6000	20000	6x	75B	105B	34	**3.142**	**38.6**	**47.6**	**26.4**	–
Dense Baseline	270M	–	100000	–	–	209B	170	3.092	40.2	47.5	**26.2**	–
Dense TST	270M	30000	100000	6x	377B	524B	170	**3.048**	**42.6**	**50.3**	25.5	–
Dense Baseline	600M	–	20000	–	–	42B	61	3.019	43.5	51.7	25.5	–
Dense TST	600M	6000	20000	6x	75B	105B	61	**2.943**	**48.2**	**52.5**	**26.9**	–
Dense Baseline	3B	–	20000	–	–	42B	**247**	2.808	57.6	60.6	31.9	31.2
Dense Baseline	3B	–	36000	–	–	75B	443	2.677	62.3	65.9	34.9	32.7
Dense Baseline	3B	–	50000	–	–	105B	622	**2.640**	**63.9**	**67.3**	**36.8**	**33.3**
Dense TST	3B	6000	20000	6x	75B	105B	**247**	2.676	62.4	66.3	36.0	32.8
MoE Baseline	10B A1B	–	125000	–	–	1.05T	12311	2.252	70.1	73.8	46.3	37.4
MoE TST	10B A1B	12483	49983	16x	1.68T	2T	**4768**	**2.236**	**71.2**	**74.2**	**47.3**	**39.0**

Table 1: Overview of TST's advantage across different configurations compared to standard training (baseline). The TST Tokens denote raw data token count before compression. All evals are 0-shot. More results are shown in Table 3, Appendix E.

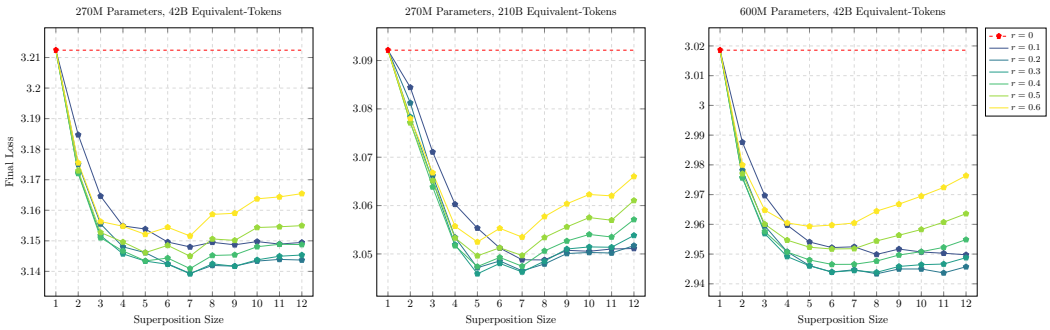


Figure 4: Superposition results with respect to loss at varying superposition bag sizes and superposition step ratio r, where r is the ratio of number of steps trained in the superposition regime, with $1 - r$ trained in the normal regime. Each data point is a fully trained model using TST first and converted to a standard AR LM. The full set of results are attached in Appendix E.

For the large 10B Mixture-of-Experts scaling validation run, we broadly follow the Qwen3 [57] training procedures, and use the Qwen3 architecture, but scaled down to match the size of a 10B total, 1B active parameter model trained to 1.05T tokens. A 50/50 mix of FineWeb-Edu [39] and DCLM was used for this larger training run, along with a constant batch size of 8M tokens per step. The TST variant was trained with the settings shown in Table 1, for a total of 2T data tokens.

For the optimizer, we use AdamW [38], with the optimal learning rate for the 270M and 600M models found using a sweep shown in Figure 7, Appendix B. For the 3B and 10B models, we use the recommended learning rate of 2×10^{-4} and 3×10^{-4} respectively, as a full sweep is too expensive at this scale. For all training runs, we use the standard $\beta_1 = 0.9$ and $\beta_2 = 0.95$, along with the Warmup-Stable-Decay [24, 56] learning rate scheduler. We warm up for a constant 2000 steps and then decay for the last 10% steps.

We run the standard set of LLM evaluations at the final step checkpoint after the recovery phase, including ARC [11], BoolQ [10], HellaSwag [60], MMLU [22], OpenBookQA [42], PIQA [7] and Winogrande [47]. All evaluations are performed using the Eleuther AI LM-Eval harness [17], using a 0-shot prompting setting. These experiments are illustrated in Table 1 and Figure 4 highlights the robustness and significant benefit of using TST, as it outperforms the baseline in all equal-FLOPs or equal-loss settings tested.

5 Discussion

5.1 On Comparisons to Auxiliary-loss Methods

One may wonder about how TST compares empirically to multi-token prediction (MTP) [20] and related auxiliary-loss methods [34, 37, 41, 64]. We argue that such comparisons are not directly

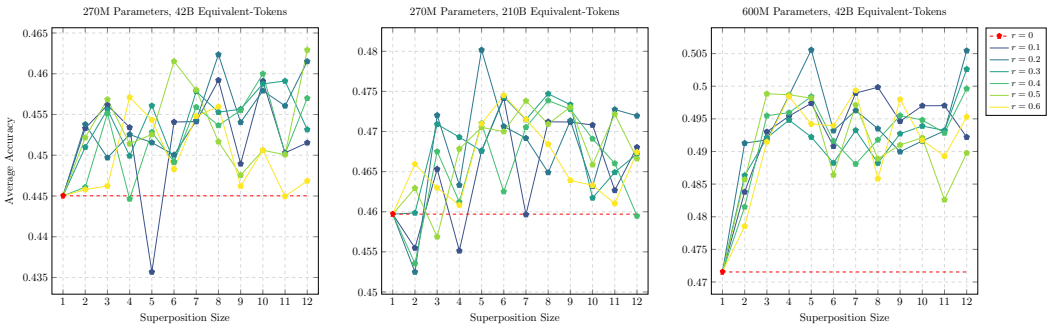


Figure 5: Downstream evals at varying superposition bag sizes and superposition step ratio r, where the average is the arithmetic average of (arc-c, arc-e, boolq, hellaswag, openbookqa, piqa, winogrande). The full set of results are attached in Appendix E.

meaningful for our setting. MTP and its variants do not increase training-time throughput: they process the same number of tokens per FLOP as the baseline, while adding parameters and an auxiliary loss term. Additionally, in the case of MTP and some of the related methods, emphasis is placed on inference-time gains via speculative decoding, an aspect that is not addressed in this work. TST occupies a different point in the design space: it strictly increases tokens-per-FLOP during training, leaves the inference-time architecture untouched, and we observe consistent gains from 270M to 10B parameters. We therefore view TST as orthogonal to, rather than competing with, auxiliary-loss methods, and combining the two is a natural direction for future work.

5.2 Input and Output Superposition

We perform ablations adding the input-only, output-only and full superposition settings with a superposition bag size $s = 4$ for a ratio $r = 0.5$ of the total steps. The input-only method only bags the input token embeddings but only predicts one next token, and the output-only method processes individual tokens in the input but predicts the next bag of tokens in the output.

Figure 6 illustrates that all superposition settings outperform the baseline, but emphasizes that input or output superposition alone does not capture the complete improvement of full superposition, and the combination of both yields a further gain without signs of interference. We understand this as evidence that TST is not a single trick but two orthogonal mechanisms: the input superposition changes the input granularity and FLOPs cost per unit of information, while the output side modifies the prediction target and the gradients.

Output Superposition Next bag-of-tokens prediction is using bag-of-words summaries of the future, reminiscent of seminal works in word vector representation [43], extractive summarization [40] and information retrieval [51], repurposed as a local supervision signal for an autoregressive model. The closest work in the literature is *future summary prediction* [41], which also targets a compressed view of the future. The important difference is architectural rather than conceptual: the authors attach an auxiliary head with a binary cross-entropy loss on top of the regular next-token objective, paying extra parameters and an extra loss term. We keep a single cross-entropy loss on the single main head and only replace the target. An even closer match, to our knowledge not in the peer-reviewed literature, is a `modded-nanogpt` speedrun entry [27] that concurrently proposed a next-bag-of-tokens loss. It differs from ours in two design choices: exponential weighting of the bag, and a smooth interpolation into next-token prediction rather than a hard switch.

Output Bag Weighting Our experiments show U-shaped loss plots when varying the superposition bag size, highlighting the need to correctly tune this hyperparameter to find the optimal setting, with the risk of slightly suboptimal results if overshooting. We tried different weighting schemes to average the loss participation of each term in the bag-of-tokens, detailed in Appendix D. The power-law weighting scheme was the most promising at large superposition sizes, so we performed a new set of experiments and compared it to the uniform average and reported the results in Figure 8,

Appendix D. The power-law weighted average results in higher loss than the uniform average for smaller superposition sizes, but outperforms it and is more stable for $s \geq 8$.

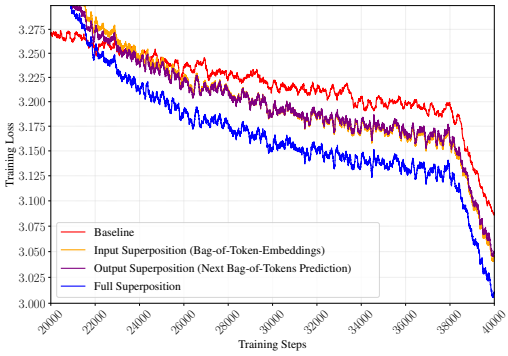


Figure 6: Input and Output Superposition ablations, only the recovery phase (ii) is represented.

Input Superposition Input superposition has, as far as we know, little direct analog in the LLM pretraining literature. The underlying reason for its success is an open question. One interpretation we find plausible is that the first phase acts as a form of pre-pre-training as studied by Hu et al. [23] and Lee et al. [30]: before learning full-resolution language, the model is exposed to a simpler distribution that already shares coarse statistical structure with natural language (*e.g.* local topic, co-occurrence), and carries that inductive prior into phase (ii).

A second, not exclusive, interpretation is that averaging in embedding space implicitly regularizes the embedding geometry, since many random s-grams must remain linearly separable once summed. We do not have the interpretability evidence to pick between these.

Beyond language models, training first on a lower-resolution, higher-throughput version of the data and then on the full-resolution version is an old idea that recurs in recent work: Anagnostidis et al. [4] schedule patch size from coarse to fine for Vision Transformers, Minixhofer et al. [44] start from a pretrained subword-LLM that is then converted into a model capable of processing finer-grained UTF-8 bytes.

Input superposition can be read as the same principle applied to token embeddings. We see this as evidence that the coarse-to-fine granularity schedule is a reusable ingredient of efficient pretraining recipes, rather than a property of any specific modality.

5.3 Two-Phase Task Alignment

The natural question arises: why has this effect not been observed in numerous prior work that resembles TST? Much of the existing literature on multi-stage pre-training and transfer learning relies on an **alignment phase**, in which the main model is frozen and only a small adapter is trained before unfreezing everything the recovery phase. We argue that this design choice is precisely what has obscured the effect we exploit in TST.

We hypothesize that the internal circuitry of a LLM is highly sensitive to its input and output representations. TST is, to our knowledge, one of the few compressive LLM pre-training method in which the input embedding and output LM head are shared without modification across the superposition and recovery phases, avoiding the representational mismatch that previous methods incur (and that adapter-based alignment is designed to patch over).

To test this hypothesis, we run a Dense TST 3B experiment matching the setup in Table 1, but we randomly re-initialize the input embedding and output LM head at the start of the recovery phase. As shown in Table 2, perturbing the input/output representations between the two phases completely eliminates the gains of TST, and makes it even worse than the baseline training where the TST steps are completely wasted and do not contribute to the final model. Although not definitive, this result supports our hypothesis that representation alignment across the two phases is a key reason why TST succeeds where prior compressive approaches have required explicit alignment training.

Model	Params	TST Steps	Total Steps	TST Bag Size	TST Tokens	Total Tokens	B200-Hours (↓)	Final Loss (↓)
Dense Baseline	3B	–	20000	–	–	42B	247	<u>2.808</u>
Dense TST	3B	6000	20000	6x	75B	105B	247	**2.676**
Dense TST w/ Randomization	3B	6000	20000	6x	75B	105B	247	2.938

Table 2: Comparison between TST and TST with randomization where we re-initialize the input embedding and LM head layers between the superposition phase and the recovery phase.

6 Conclusion

In this work, we describe a high throughput training paradigm for LLMs: Token Superposition Training. During the token superposition regime, sample throughput is increased s-fold without changing per-step FLOPs, parallelism, model architecture, tokenizer, or data. The following recovery phase is a return to the standard LLM pretraining regime, exhibiting a fast recovery period, quickly outperforming the loss of an equal-FLOPs baseline pretraining. TST significantly increases the pretraining efficiency compared to baseline pretraining at the same computational cost (*c.f.* Figure 3a). Alternatively, the same loss can be achieved at around half the computational cost (*c.f.* Figure 3b). Overall, we find this new paradigm to be robust to hyperparameter choice within a reasonable range (superposition bag size $s \in [\![4, 8]\!]$ and step ratio $r \in [0.2, 0.4]$).

7 Limitations, Future Work and Broader Impacts

TST effectively trades more data consumption for better loss at a given computational cost. The underlying assumption is that LLM pretraining is performed under compute-bound constraints rather than data-bound constraints. Kim et al. [28] recently argued that this assumption will be wrong in the future, given current trends. In this alternative view, output-only superposition offers a significant advantage, as it outperforms the baseline pretraining regime without increasing data consumption. We leave the study of these settings and the comparison with auxiliary loss methods, such as MTP, for future work.

Folding the initial sequence into a sequence of bags of s tokens results in a longer effective context during TST compared to the baseline regime. This could likely have positive effects on long context performance that we did not evaluate, as there would be less truncation or splitting of native long context data, leaving this to future work.

We also did not perform larger scale ablations or multiple identical runs to evaluate statistical significance due to limited compute resources. Future work could investigate scaling laws of token superposition, in order to predict the best TST settings for larger model sizes, including industry-scale pretraining.

We proposed some hypotheses on the phenomena involved in TST, but further interpretability work on this subject could definitely improve the understanding of the underlying mechanisms and the ramifications of token superposition.

Broader Impacts: Our work improves the efficiency of large language model pre-training, reducing computational cost and energy usage, which may improve accessibility to a broader range of researchers. However, increased efficiency may also accelerate the development and deployment of such models, potentially amplifying known risks including misuse for generating harmful content and concerns around bias, fairness, and privacy. While our contribution is methodological and does not directly introduce new capabilities, it may indirectly increase the scale at which these systems are trained and used. We encourage future work to pair efficiency improvements with advances in safety and responsible deployment.

References

[1] S. Agarwal, L. Ahmad, J. Ai, S. Altman, A. Applebaum, E. Arbus, R. K. Arora, Y. Bai, B. Baker, H. Bao, and others. gpt-oss-120b & gpt-oss-20b model card.

[2] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millicah, M. Reynolds, R. Ring, E. Rutherford, S. Cabi, T. Han, Z. Gong, S. Samangooei, M. Monteiro, J. Menick, S. Borgeaud, A. Brock, A. Nematzadeh, S. Sharifzadeh, M. Binkowski, R. Barreira, O. Vinyals, A. Zisserman, and K. Simonyan. Flamingo: a visual language model for few-shot learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, pages 23716–23736, Red Hook, NY, USA, Nov. 2022. Curran Associates Inc. ISBN 978-1-7138-7108-8.

[3] L. B. Allal, A. Lozhkov, E. Bakouch, G. M. Blázquez, G. Penedo, L. Tunstall, A. Marafioti, H. Kydlíček, A. P. Lajarín, V. Srivastav, J. Lochner, C. Fahlgren, X.-S. Nguyen, C. Fourrier, B. Burtenshaw, H. Larcher, H. Zhao, C. Zakka, M. Morlon, C. Raffel, L. v. Werra, and T. Wolf. SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model, Feb. 2025. URL http://arxiv.org/abs/2502.02737. arXiv:2502.02737 [cs].

[4] S. Anagnostidis, G. Bachmann, I. Schlag, and T. Hofmann. Navigating scaling laws: compute optimality in adaptive model training. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 1511–1530, Vienna, Austria, 2024. JMLR.org.

[5] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, and others. Qwen technical report.

[6] X. Bi, D. Chen, G. Chen, S. Chen, D. Dai, C. Deng, H. Ding, K. Dong, Q. Du, Z. Fu, and others. Deepseek llm: Scaling open-source language models with longtermism.

[7] Y. Bisk, R. Zellers, J. Gao, Y. Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[8] M. Chen, B. Hui, Z. Cui, J. Yang, D. Liu, J. Sun, J. Lin, and Z. Liu. Parallel Scaling Law for Language Models. Oct. 2025. URL https://openreview.net/forum?id=dEi1S731lk.

[9] X. Cheng, W. Zeng, D. Dai, Q. Chen, B. Wang, Z. Xie, K. Huang, X. Yu, Z. Hao, Y. Li, and others. Conditional memory via scalable lookup: A new axis of sparsity for large language models.

[10] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. 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)*, pages 2924–2936, 2019.

[11] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

[12] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[13] D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang. DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.70. URL https://aclanthology.org/2024.acl-long.70/.

[14] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. Oct. 2020. URL https://openreview.net/forum?id=YicbFdNTTy&utm_campaign=The%20Batch&utm_source=hs_email&utm_medium=email&_hsenc=p2ANqtz--bIpWoAA0d8Ugha6WmwlzJEFeLwluYNZSx-7AAH9r5Kdq3UTcUJwY1X4RnbL0IOgx_32-d.

[15] W. Ebeling and T. Pöschel. Entropy and Long-Range Correlations in Literary English. *Europhysics Letters*, 26(4):241, May 1994. ISSN 0295-5075. doi: 10.1209/0295-5075/26/4/001. URL https://doi.org/10.1209/0295-5075/26/4/001.

[16] S. Y. Gadre, G. Smyrnis, V. Shankar, S. Gururangan, M. Wortsman, R. Shao, J. Mercat, A. Fang, J. Li, S. Keh, and others. Language models scale reliably with over-training and on downstream tasks.

[17] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. A framework for few-shot language model evaluation, 12 2023. URL https://zenodo.org/records/10256836.

[18] T. Gigant, B. Peng, and J. Quesnelle. Decoupling the Benefits of Subword Tokenization for Language Model Training via Byte-level Simulation, Apr. 2026. URL http://arxiv.org/abs/2604.27263. arXiv:2604.27263 [cs].

[19] H. Gisserot-Boukhlef, N. Boizard, M. Faysse, D. M. Alves, E. Malherbe, A. Martins, C. Hudelot, and P. Colombo. Should We Still Pretrain Encoders with Masked Language Modeling? Oct. 2025. URL https://openreview.net/forum?id=jpz7e3jhRq.

[20] F. Gloeckle, B. Y. Idrissi, B. Rozière, D. Lopez-Paz, and G. Synnaeve. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 15706–15734, Vienna, Austria, 2024. JMLR.org.

[21] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, and others. The llama 3 herd of models.

[22] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[23] M. Y. Hu, J. Petty, C. Shi, W. Merrill, and T. Linzen. Between Circuits and Chomsky: Pre-pretraining on Formal Languages Imparts Linguistic Biases. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9691–9709, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.478. URL https://aclanthology.org/2025.acl-long.478/.

[24] S. Hu, Y. Tu, X. Han, C. He, G. Cui, X. Long, Z. Zheng, Y. Fang, Y. Huang, W. Zhao, X. Zhang, Z. L. Thai, K. Zhang, C. Wang, Y. Yao, C. Zhao, J. Zhou, J. Cai, Z. Zhai, N. Ding, C. Jia, G. Zeng, D. Li, Z. Liu, and M. Sun. Minicpm: Unveiling the potential of small language models with scalable training strategies, 2024. URL https://arxiv.org/abs/2404.06395.

[25] S. Hwang, B. Wang, and A. Gu. Dynamic Chunking for End-to-End Hierarchical Sequence Modeling, July 2025. URL http://arxiv.org/abs/2507.07955. arXiv:2507.07955 [cs].

[26] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mixtral of Experts, Jan. 2024. URL http://arxiv.org/abs/2401.04088. arXiv:2401.04088 [cs].

[27] KellerJordan. New Record: Multi-token prediction and Untie LM Head 2/3rds through training (119.76 seconds) by varunneal · Pull Request #178 · KellerJordan/modded-nanogpt. URL https://github.com/KellerJordan/modded-nanogpt/pull/178.

[28] K. Kim, S. Kotha, P. Liang, and T. Hashimoto. Pre-training under infinite compute, Sept. 2025. URL http://arxiv.org/abs/2509.14786. arXiv:2509.14786 [cs].

[29] T. Kudo and J. Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 conference on empirical methods in natural language processing: System demonstrations*, pages 66–71.

[30] D. Lee, S. Han, A. Kumar, and P. Agrawal. Training Language Models via Neural Cellular Automata, 2026. URL https://arxiv.org/abs/2603.10055. Version Number: 1.

[31] Y. Leviathan, M. Kalman, and Y. Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *ICML'23*, pages 19274–19286, Honolulu, Hawaii, USA, 2023. JMLR.org.

[32] J. Li, A. Fang, G. Smyrnis, M. Ivgi, M. Jordan, S. Gadre, H. Bansal, E. Guha, S. Keh, K. Arora, S. Garg, R. Xin, N. Muennighoff, R. Heckel, J. Mercat, M. Chen, S. Gururangan, M. Wortsman, A. Albalak, Y. Bitton, M. Nezhurina, A. Abbas, C.-Y. Hsieh, D. Ghosh, J. Gardner, M. Kilian, H. Zhang, R. Shao, S. Pratt, S. Sanyal, G. Ilharco, G. Daras, K. Marathe, A. Gokaslan, J. Zhang, K. Chandu, T. Nguyen, I. Vasiljevic, S. Kakade, S. Song, S. Sanghavi, F. Faghri, S. Oh, L. Zettlemoyer, K. Lo, A. El-Nouby, H. Pouransari, A. Toshev, S. Wang, D. Groeneveld, L. Soldaini, P. W. Koh, J. Jitsev, T. Kollar, A. G. Dimakis, Y. Carmon, A. Dave, L. Schmidt, and V. Shankar. Datacomp-lm: In search of the next generation of training sets for language models, 2024.

[33] W. Liang, T. Liu, L. Wright, W. Constable, A. Gu, C.-C. Huang, I. Zhang, W. Feng, H. Huang, J. Wang, S. Purandare, G. Nadathur, and S. Idreos. TorchTitan: One-stop PyTorch native solution for production ready LLM pretraining. Oct. 2024. URL https://openreview.net/forum?id=SFN6Wm7YBI.

[34] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, and others. Deepseek-v3 technical report. .

[35] A. Liu, J. Hayase, V. Hofmann, S. Oh, N. A. Smith, and Y. Choi. SuperBPE: Space travel for language models. .

[36] H. Liu, J. Zhang, C. Wang, X. Hu, L. Lyu, J. Sun, X. Yang, B. Wang, F. Li, Y. Qian, and others. Scaling embeddings outperforms scaling experts in language models. .

[37] Y. Liu, Y. Song, Y. Wang, K. Ge, A. Lamb, Q. Guo, K. Chen, B. Zhou, and Z. Lin. Next Concept Prediction in Discrete Latent Space Leads to Stronger Language Models, Feb. 2026. URL http://arxiv.org/abs/2602.08984. arXiv:2602.08984 [cs].

[38] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[39] A. Lozhkov, L. Ben Allal, L. von Werra, and T. Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

[40] H. P. Luhn. The Automatic Creation of Literature Abstracts. *IBM Journal of Research and Development*, 2(2):159–165, Apr. 1958. ISSN 0018-8646. doi: 10.1147/rd.22.0159. URL https://ieeexplore.ieee.org/document/5392672. Conference Name: IBM Journal of Research and Development.

[41] D. Mahajan, S. Goyal, B. Y. Idrissi, M. Pezeshki, I. Mitliagkas, D. Lopez-Paz, and K. Ahuja. Beyond Multi-Token Prediction: Pretraining LLMs with Future Summaries, Oct. 2025. URL http://arxiv.org/abs/2510.14751. arXiv:2510.14751 [cs].

[42] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2381–2391, 2018.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. Jan. 2013. URL https://www.semanticscholar.org/paper/Efficient-Estimation-of-Word-Representations-in-Mikolov-Chen/f6b51c8753a871dc94ff32152c00c01e94f90f09.

[44] B. Minixhofer, T. Murray, T. Limisiewicz, A. Korhonen, L. Zettlemoyer, N. A. Smith, E. M. Ponti, L. Soldaini, and V. Hofmann. Bolmo: Byteifying the Next Generation of Language Models, Dec. 2025. URL http://arxiv.org/abs/2512.15586. arXiv:2512.15586 [cs].

[45] S. Nie, F. Zhu, Z. You, X. Zhang, J. Ou, J. Hu, J. Zhou, Y. Lin, J.-R. Wen, and C. Li. Large Language Diffusion Models. Oct. 2025. URL https://openreview.net/forum?id=KnqiC0znVF.

[46] A. Pagnoni, R. Pasunuru, P. Rodriguez, J. Nguyen, B. Muller, M. Li, C. Zhou, L. Yu, J. E. Weston, L. Zettlemoyer, G. Ghosh, M. Lewis, A. Holtzman, and S. Iyer. Byte Latent Transformer: Patches Scale Better Than Tokens. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9238–9258, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.453. URL https://aclanthology.org/2025.acl-long.453/.

[47] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[48] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 1715–1725.

[49] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[50] C. Shao, F. Meng, and J. Zhou. Beyond next token prediction: Patch-level training for large language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=dDpB23VbVa.

[51] K. Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[52] Y. Tay, M. Dehghani, V. Q. Tran, X. Garcia, J. Wei, X. Wang, H. W. Chung, D. Bahri, T. Schuster, S. Zheng, D. Zhou, N. Houlsby, and D. Metzler. UL2: Unifying Language Learning Paradigms. Sept. 2022. URL https://openreview.net/forum?id=6ruVLB727MC.

[53] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, and others. Llama: Open and efficient foundation language models.

[54] M. Videau, B. Y. Idrissi, A. Leite, M. Schoenauer, O. Teytaud, and D. Lopez-Paz. From Bytes to Ideas: Language Modeling with Autoregressive U-Nets. Oct. 2025. URL https://openreview.net/forum?id=FnFf7Ru2ur.

[55] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Oct. 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J&trk=public_post_comment-text.

[56] K. Wen, Z. Li, J. Wang, D. Hall, P. Liang, and T. Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective, 2024. URL https://arxiv.org/abs/2410.05192.

[57] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, and others. Qwen3 technical report.

[58] J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, Y. Wei, L. Wang, Z. Xiao, Y. Wang, C. Ruan, M. Zhang, W. Liang, and W. Zeng. Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention. In W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, editors, *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1126. URL https://aclanthology.org/2025.acl-long.1126/.

[59] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed. Big bird: transformers for longer sequences. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pages 17283–17297, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 978-1-7138-2954-6. URL https://dl.acm.org/doi/10.5555/3495724.3497174.

[60] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 4791–4800, 2019.

[61] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proc. VLDB Endow.*, 16(12):3848–3860, 2023. ISSN 2150-8097. doi: 10.14778/3611540.3611569. URL https://dl.acm.org/doi/10.14778/3611540.3611569.

[62] L. Zheng, X. Li, Q. Liu, X. Feng, and L. Kong. Proxy Compression for Language Modeling, Feb. 2026. URL http://arxiv.org/abs/2602.04289. arXiv:2602.04289 [cs].

[63] R.-J. Zhu, Z. Wang, K. Hua, T. Zhang, Z. Li, H. Que, B. Wei, Z. Wen, F. Yin, H. Xing, and others. Scaling latent reasoning via looped language models.

[64] Z. M. K. Zuhri, E. H. Fuadi, and A. F. Aji. Predicting the Order of Upcoming Tokens Improves Language Modeling, Feb. 2026. URL http://arxiv.org/abs/2508.19228. arXiv:2508.19228 [cs].

A Code

```

[...] # within train loop

if superposition_bag_size is not None and superposition_bag_size > 1:
    bs, seq = inputs.shape
    inputs = inputs.reshape(bs, seq//superposition_bag_size,
    superposition_bag_size)
```

Listing 1: Input folding in Pytorch

```

[...] # within model forward

            # Using superposition
            if len(tokens.shape) == 3:
                bs, sp_seq, superposition_bag_size = tokens.shape

                # Sum in float32 for better numerical precision
                h = self.tok_embeddings(tokens[..., 0])
                h_dtype = h.dtype
                h = h.float()
                for i in range(1, superposition_bag_size):
                    h = h + self.tok_embeddings(tokens[..., i]).float()

                h = (h / superposition_bag_size).to(h_dtype)

            else:
                h = self.tok_embeddings(tokens)
```

Listing 2: Bag-of-Token embeddings input in Pytorch

```
import torch

def cross_entropy_loss(pred: torch.Tensor, labels: torch.Tensor) -> torch.
    Tensor:

    # Compute shapes
    bs, seq, dim = pred.shape
    label_bs, label_seq = labels.shape
    superposition_bag_size = label_seq // seq
    superposition_offset = superposition_bag_size - 1

    # Pre-flatten and perform causal padding
    pred = pred.flatten(0, 1).float()
    labels = torch.nn.functional.pad(labels, (0, superposition_offset), mode=
    'constant', value=-100)[..., superposition_offset:].view((bs, seq,
    superposition_bag_size))

    # Compute loss
    loss = 0.
    w_total = 0.
    for i in range(superposition_bag_size):
        w = 1 # uniform weighting
        target = labels[..., i].flatten(0, 1)
        loss += w * torch.nn.functional.cross_entropy(pred, target)
        w_total += w

    return loss / w_total
```

Listing 3: Next bag-of-words prediction loss code in Pytorch

B Learning rate sweeps

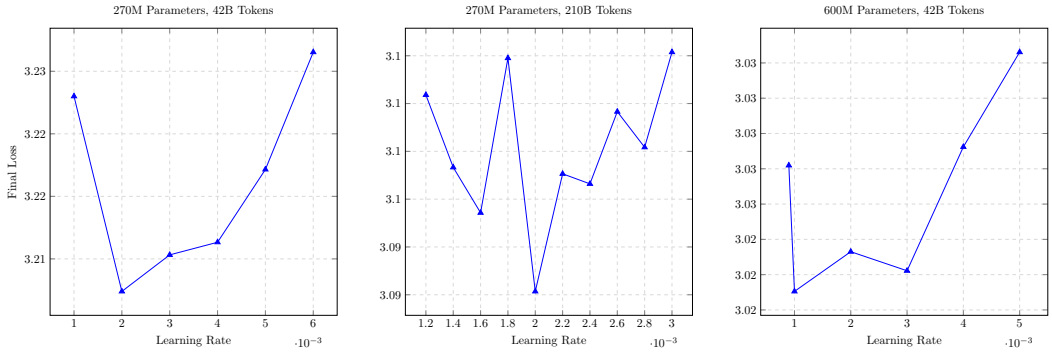


Figure 7: Learning rate sweeps at varying model sizes, with the optimal learning rate being used for all the training runs. The final lr used in order from left to right are 2×10^{-3}, 2×10^{-3} and 1×10^{-3}.

C Loss Derivations

Setup. Given logits $\mathbf{z} \in \mathbb{R}^V$, let $Z = \sum_{i=1}^{V} \exp(z_i)$ and write the softmax probability as $P(i) = \exp(z_i)/Z$. For a target distribution $\mathbf{t}$ over the vocabulary, cross-entropy and KL divergence are

$$\mathrm{CE}(\mathbf{t}, P) = -\sum_i t_i \log P(i), \qquad \mathrm{KL}(\mathbf{t} \,\|\, P) = \mathrm{CE}(\mathbf{t}, P) - H(\mathbf{t}),$$

where $\mathbf{t}$ is the target probability vector, and where $H(\mathbf{t}) = -\sum_i t_i \log t_i$. Cross-entropy vanishes at its minimum only when $H(\mathbf{t}) = 0$, i.e. when $\mathbf{t}$ is one-hot. KL divergence always vanishes at $P = \mathbf{t}$ regardless of the target.

We now consider a bag $\mathbf{y}$ of s valid target tokens, with two variants of a multi-hot cross-entropy (MCE) loss for a bag of s valid target tokens $\mathbf{y}$.

C.1 MCE: equal-probability targets

The MCE loss used throughout the paper treats the bag as a multi-hot target and pushes each valid token's probability towards $1/s$, so that the bag tokens try to end up with an equal probability. Therefore, the target is the uniform distribution over the bag:

$$t_y = \begin{cases} 1/s & y \in \mathbf{y} \\ 0 & \text{otherwise} \end{cases}$$

Standard cross-entropy with this new target is

$$\mathrm{CE}(\mathbf{t}, P) = -\frac{1}{s} \sum_{y \in \mathbf{y}} \log \frac{\exp(z_y)}{\sum_{i=1}^{V} \exp(z_i)} = -\frac{1}{s} \sum_{y \in \mathbf{y}} \log P(y)$$

Unlike the one-hot case, this target has nonzero entropy $H(\mathbf{t}) = \log s$, so plain CE bottoms out at $\log s$ rather than 0. To recover the same "vanishes at the optimum" behaviour as standard CE, we

subtract the entropy of the target (i.e. we use KL divergence):

$$\begin{aligned}
\mathcal{L}_{\text{MCE}}(\mathbf{z}, \mathbf{y}) &= \text{KL}(\mathbf{t} \,\|\, P) \\
&= -\frac{1}{s} \sum_{y \in \mathbf{y}} \log P(y) - \log s \\
&= -\frac{1}{s} \sum_{y \in \mathbf{y}} \log \frac{s \, \exp(z_y)}{\sum_{i=1}^{V} \exp(z_i)} \\
&= -\frac{1}{s} \sum_{y \in \mathbf{y}} \left(z_y - \log \sum_{i=1}^{V} \exp(z_i) + \log s \right) \\
&= -\frac{1}{s} \left(\sum_{y \in \mathbf{y}} z_y - s \log \sum_{i=1}^{V} \exp(z_i) + s \log s \right) \\
&= -\frac{1}{s} \sum_{y \in \mathbf{y}} z_y + \log \sum_{i=1}^{V} \exp(z_i) - \log s
\end{aligned} \tag{4}$$

Rearranging the terms, we get:

$$\begin{aligned}
\mathcal{L}_{\text{MCE}}(\mathbf{z}, \mathbf{y}) &= -\frac{1}{|\mathbf{y}|} \sum_{y \in \mathbf{y}} z_y + \log \sum_{i=1}^{V} \exp(z_i) - \log |\mathbf{y}| \\
&= \frac{1}{|\mathbf{y}|} \sum_{y \in \mathbf{y}} \left(-z_y + \log \sum_{i=1}^{V} \exp(z_i) \right) - \log |\mathbf{y}| \\
&= \frac{1}{|\mathbf{y}|} \sum_{y \in \mathbf{y}} \mathcal{L}_{\text{CE}}(\mathbf{z}, y) - \log |\mathbf{y}|
\end{aligned} \tag{5}$$

C.2 MCE_{Alt}: sum-to-one probability targets

We also investigated a different formulation of the MCE loss we used, where we target the sum of the probabilities of all valid labels to be 1 instead of targeting them to be equal-probability. Here we do not pick a single target distribution; instead we only require that the total probability mass on the bag be 1. This effectively lets the model choose its own weighting across bag tokens. A natural way to express this is to treat the bag as a single composite label with probability

$$P(\mathbf{y}) = \sum_{y \in \mathbf{y}} P(y) = \frac{\sum_{y \in \mathbf{y}} \exp(z_y)}{\sum_{i=1}^{V} \exp(z_i)}.$$

Cross-entropy against a one-hot target on this composite label is

$$\begin{aligned}
\mathcal{L}_{\text{MCE}_{\text{Alt}}}(\mathbf{z}, \mathbf{y}) &= -\log P(\mathbf{y}) \\
&= -\log \frac{\sum_{y \in \mathbf{y}} \exp(z_y)}{\sum_{i=1}^{V} \exp(z_i)} \\
&= -\log \sum_{y \in \mathbf{y}} \exp(z_y) + \log \sum_{i=1}^{V} \exp(z_i)
\end{aligned} \tag{6}$$

Because the implicit target is a one-hot over composite labels (all mass in "the bag"), its entropy is zero and no correction is needed: the loss vanishes exactly when $\sum_{y \in \mathbf{y}} P(y) = 1$, just like ordinary CE on a single label.

In all the limited small-scale experiments we have done with MCE_{Alt}, it seemed to perform identically[2] compared to the equal-probability loss of MCE. We therefore did not explore it further, since unlike MCE it does not reduce to a simplified form that uses CE, it would require a custom loss function, reduce the training speed, increase memory usage, and add unnecessary complexity to the final method. We leave a more thorough exploration of this variant for future work.

[2]We trained models using one variant of MCE or the other, and we obtained the same final loss after the recovery phase if everything else is kept identical.

D Non-uniform Multi-hot Cross-Entropy

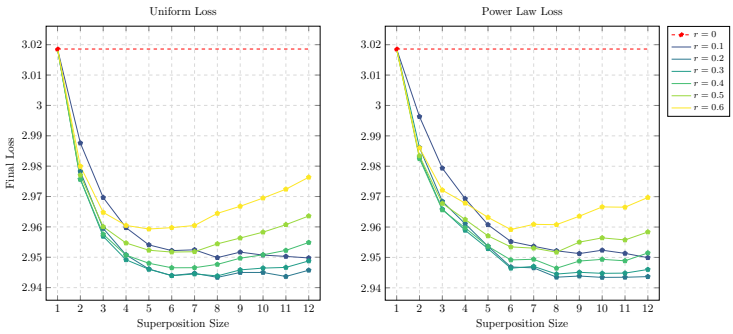
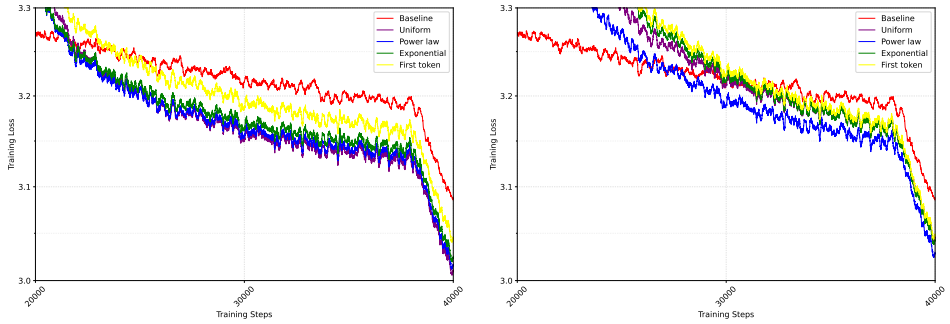


Figure 8: Comparison between superposition using uniform output loss and power law output loss at varying superposition window sizes and superposition ratio r, with a 600M-parameter model trained at 42B tokens. The final loss is the final training loss using standard next-token Cross-Entropy Loss.

For other multi-hot target distributions, we consider functions g that are monotonically decreasing with the position i in the bag.

$$\mathcal{L}_{\text{MCE}}(\mathbf{z}, \mathbf{y}, g) = \frac{1}{\sum_i g(i)} \sum_{y \in \mathbf{y}} g(i) \mathcal{L}_{\text{CE}}(\mathbf{z}, y) \tag{7}$$

- Uniform: $i \mapsto 1$
- Power law: $i \mapsto \frac{1}{i}$
- Exponential: $i \mapsto \exp(-i)$, following [27]
- First token: $i \mapsto \delta_1(i)$



(a) Superposition bag size $s = 4$

(b) Superposition bag size $s = 16$

Figure 9: Training loss after resuming from weighted superposition

At a superposition bag size $s = 16$, the best setting involves a power-law distribution. However, at $s = 4$, the uniform distribution works better. We did not find a distribution that would work best for all superposition ratios.

However, we believe that the power law distribution of losses is reminiscent of the prediction difficulty of future tokens based on current context. Ebeling and Pöschel [15] investigated mutual information between pairs of letters in literary English texts, which they found to decay with distance following a power law. Inspired by this, we compute mutual information between pairs of tokens sampled from texts in the DCLM dataset and fit a power law to modelize its decay. This mutual information decay between pairs of tokens in the DCLM dataset, and a fitted power law, are illustrated in Figure 10. This result correlates with our observation for relative weighting of tokens for next bag-of-tokens prediction with large superposition bag sizes. Using the values of the fitted

power law to weight the losses per position results in a slightly better loss than the power law tested earlier.

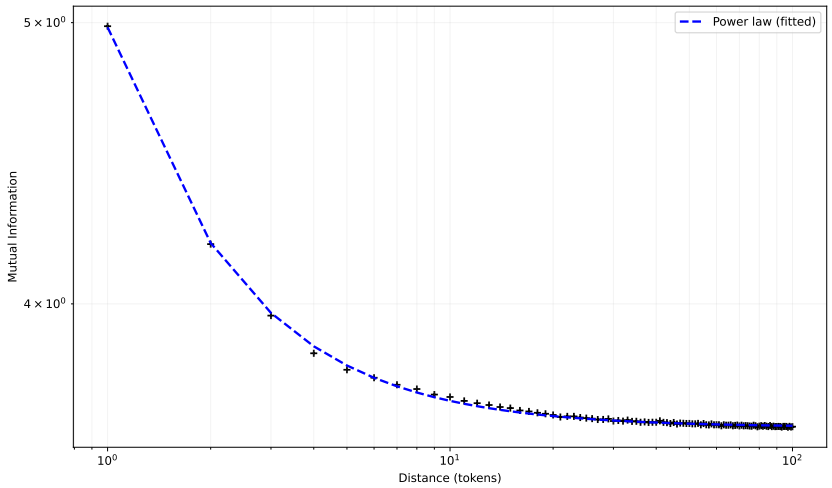


Figure 10: Mutual information between pairs of tokens in DCLM decays with distance following a power law: $d \mapsto C_0 + a * d^k$, with $C_0 \approx 3.63$, $a \approx 1.35$ and $k \approx -1.25$

E Additional Results

Unless mentioned, all evals are run with 0-shot prompting.

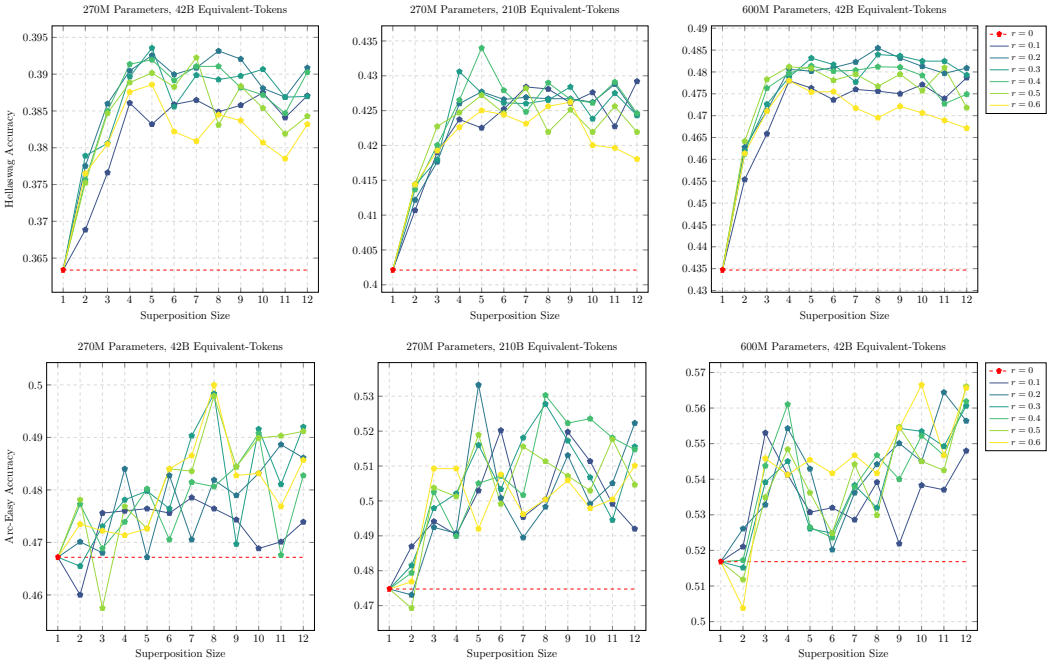


Figure 11: Rows from top to bottom: Hellaswag and ARC-Easy downstream evals at varying superposition bag sizes and superposition step ratio r.

Model	Params	TST Steps	Total Steps	B200-Hours (↓)	Final Loss (↓)	HellaSwag (↑)	ARC-E (↑)	ARC-C (↑)	MMLU (↑)	BoolQ (↑)	OpenBookQA (↑)	PIQA (↑)	Winogrande (↑)
Dense Baseline	270M	–	20000	34	3.212	36.3	46.7	24.9	–	**56.7**	29.2	66.4	**51.3**
Dense TST	270M	6000	20000	34	**3.142**	**38.6**	**47.6**	**26.4**	–	54.0	**29.8**	**67.0**	51.1
Dense Baseline	270M	–	100000	170	3.092	40.2	47.5	**26.2**	–	**58.5**	30.6	67.3	51.5
Dense TST	270M	30000	100000	170	**3.048**	**42.6**	**50.3**	25.5	–	57.9	**32.4**	**69.0**	**54.2**
Dense Baseline	600M	–	20000	61	3.019	43.5	51.7	25.5	–	**56.6**	31.2	69.0	52.6
Dense TST	600M	6000	20000	61	**2.943**	**48.2**	**52.5**	**26.9**	–	54.8	**35.0**	**70.6**	**53.9**
Dense Baseline	3B	–	20000	**247**	2.808	57.6	60.6	31.9	31.2	58.4	36.6	74.5	56.5
Dense Baseline	3B	–	36000	443	2.677	62.3	65.9	34.9	32.7	62.1	36.4	74.7	60.0
Dense Baseline	3B	–	50000	622	**2.640**	**63.9**	**67.3**	**36.8**	**33.3**	**64.6**	38.0	75.5	**60.9**
Dense TST	3B	6000	20000	**247**	2.676	62.4	66.3	36.0	32.8	60.0	**42.0**	**76.1**	59.6
MoE Baseline	10B A1B	–	125000	12311	2.252	70.1	73.8	46.3	37.4	66.2	**44.0**	77.2	61.3
MoE TST	10B A1B	12483	49983	**4768**	**2.236**	**71.2**	**74.2**	**47.3**	**39.0**	**69.4**	43.2	**77.4**	**63.0**

Table 3: Expanded results of Table 1. All evals are 0-shot.

Table 4: Final loss values with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	3.2124	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	3.1847	3.1646	3.1549	3.1539	3.1496	3.1480	3.1495	3.1487	3.1498	3.1489	3.1495	3.1505	3.1513	3.1493	3.1487
0.2	–	3.1751	3.1555	3.1480	3.1461	3.1425	3.1393	3.1419	3.1417	3.1434	3.1439	3.1437	3.1450	3.1451	3.1431	3.1455
0.3	–	3.1723	3.1517	3.1457	3.1434	3.1423	3.1392	3.1425	3.1417	3.1438	3.1450	3.1454	3.1464	3.1475	3.1456	3.1484
0.4	–	3.1720	3.1510	3.1467	3.1434	3.1444	3.1409	3.1452	3.1454	3.1481	3.1489	3.1488	3.1504	3.1520	3.1508	3.1534
0.5	–	3.1729	3.1527	3.1496	3.1461	3.1485	3.1450	3.1506	3.1502	3.1544	3.1546	3.1550	3.1574	3.1597	3.1583	3.1611
0.6	–	3.1755	3.1563	3.1548	3.1521	3.1545	3.1516	3.1587	3.1590	3.1638	3.1644	3.1655	3.1675	3.1708	3.1699	3.1734
0.7	–	3.1807	3.1635	3.1630	3.1621	3.1664	3.1635	3.1720	3.1730	3.1794	3.1806	3.1823	3.1847	3.1893	3.1882	3.1922
0.8	–	3.1904	3.1768	3.1774	3.1808	3.1879	3.1862	3.1970	3.1998	3.2075	3.2105	3.2138	3.2176	3.2239	3.2226	3.2280
0.9	–	3.2131	3.2110	3.2221	3.2324	3.2487	3.2501	3.2681	3.2765	3.2894	3.2985	3.3073	3.3146	3.3275	3.3254	3.3381
1.0	–	4.6099	5.2482	5.6289	5.8658	6.0365	6.1567	6.2477	6.3146	6.3677	6.4152	6.4529	6.4835	6.5082	6.5314	6.5519

Table 5: Final loss values with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	3.0921	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	3.0844	3.0711	3.0603	3.0553	3.0512	3.0488	3.0488	3.0508	3.0506	3.0510	3.0511	3.0525
0.2	–	3.0812	3.0662	3.0534	3.0472	3.0486	3.0465	3.0479	3.0501	3.0503	3.0502	3.0517	3.0524
0.3	–	3.0783	3.0652	3.0519	3.0459	3.0480	3.0463	3.0485	3.0510	3.0515	3.0514	3.0538	3.0551
0.4	–	3.0772	3.0639	3.0517	3.0474	3.0493	3.0475	3.0506	3.0527	3.0540	3.0535	3.0571	3.0588
0.5	–	3.0771	3.0652	3.0533	3.0496	3.0513	3.0497	3.0534	3.0556	3.0575	3.0570	3.0611	3.0627
0.6	–	3.0780	3.0668	3.0557	3.0525	3.0553	3.0535	3.0577	3.0604	3.0623	3.0620	3.0660	3.0681
0.7	–	3.0805	3.0703	3.0601	3.0574	3.0608	3.0597	3.0642	3.0673	3.0697	3.0694	3.0738	3.0762
0.8	–	3.0852	3.0770	3.0683	3.0670	3.0711	3.0706	3.0761	3.0800	3.0827	3.0830	3.0881	3.0909
0.9	–	3.0984	3.0963	3.0908	3.0934	3.1009	3.1020	3.1097	3.1161	3.1209	3.1236	3.1306	3.1348
1.0	–	4.5120	5.2018	5.5772	5.8255	5.9952	6.1179	6.2156	6.2891	6.3461	6.3890	6.4285	6.4628

Table 6: Final loss values with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	3.0186	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	2.9876	2.9697	2.9597	2.9541	2.9522	2.9525	2.9499	2.9517	2.9507	2.9503	2.9498	2.9514
0.2	–	2.9782	2.9595	2.9508	2.9461	2.9440	2.9447	2.9434	2.9450	2.9450	2.9437	2.9457	2.9465
0.3	–	2.9757	2.9569	2.9492	2.9461	2.9439	2.9445	2.9439	2.9458	2.9464	2.9467	2.9488	2.9501
0.4	–	2.9757	2.9576	2.9507	2.9480	2.9465	2.9466	2.9477	2.9497	2.9508	2.9523	2.9549	2.9558
0.5	–	2.9770	2.9601	2.9547	2.9523	2.9517	2.9518	2.9544	2.9563	2.9583	2.9607	2.9636	2.9647
0.6	–	2.9800	2.9648	2.9605	2.9593	2.9598	2.9605	2.9645	2.9668	2.9695	2.9724	2.9764	2.9779
0.7	–	2.9857	2.9723	2.9702	2.9708	2.9726	2.9747	2.9801	2.9832	2.9871	2.9911	2.9962	2.9987
0.8	–	2.9946	2.9865	2.9874	2.9918	2.9955	3.0000	3.0079	3.0132	3.0181	3.0243	3.0310	3.0356
0.9	–	3.0150	3.0194	3.0305	3.0441	3.0546	3.0659	3.0811	3.0915	3.1029	3.1146	3.1264	3.1366
1.0	–	4.3622	5.0131	5.4091	–	5.8477	5.9880	6.0884	6.1684	6.2315	6.2853	6.3318	6.3679

Table 7: Final loss values with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details. The TST loss used the power-law weighting as described in Appendix D.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	3.0186	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	2.9963	2.9794	2.9693	2.9608	2.9552	2.9537	2.9522	2.9513	2.9524	2.9513	2.9499	2.9497
0.2	–	2.9861	2.9685	2.9611	2.9537	2.9469	2.9466	2.9435	2.9439	2.9434	2.9435	2.9437	2.9442
0.3	–	2.9831	2.9659	2.9589	2.9529	2.9465	2.9471	2.9445	2.9451	2.9448	2.9448	2.9460	2.9471
0.4	–	2.9825	2.9656	2.9597	2.9537	2.9492	2.9494	2.9465	2.9488	2.9494	2.9489	2.9515	2.9528
0.5	–	2.9834	2.9678	2.9625	2.9571	2.9535	2.9531	2.9517	2.9550	2.9564	2.9558	2.9584	2.9606
0.6	–	2.9860	2.9721	2.9679	2.9632	2.9592	2.9609	2.9608	2.9636	2.9666	2.9665	2.9697	2.9721
0.7	–	2.9908	2.9786	2.9766	2.9733	2.9707	2.9741	2.9752	2.9781	2.9824	2.9828	2.9865	2.9895
0.8	–	2.9988	2.9904	2.9922	2.9911	2.9909	2.9967	2.9988	3.0038	3.0098	3.0111	3.0163	3.0202
0.9	–	3.0162	3.0184	3.0277	3.0334	3.0403	3.0510	3.0579	3.0672	3.0785	3.0829	3.0925	3.0997
1.0	–	3.0669	–	–	–	–	–	–	–	–	–	–	–

Table 8: Final ARC-Challenge evals with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	0.2491	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.2543	0.2509	0.2432	0.2491	0.2645	0.2517	0.2594	0.2432	0.2594	0.2526	0.2526	0.2526	0.2568	0.2440	0.2611
0.2	–	0.2500	0.2483	0.2526	0.2509	0.2457	0.2483	0.2517	0.2679	0.2560	0.2517	0.2628	0.2415	0.2509	0.2500	0.2602
0.3	–	0.2449	0.2602	0.2577	0.2534	0.2637	0.2509	0.2534	0.2568	0.2526	0.2543	0.2645	0.2577	0.2577	0.2491	0.2577
0.4	–	0.2466	0.2474	0.2398	0.2594	0.2637	0.2491	0.2483	0.2679	0.2509	0.2534	0.2739	0.2594	0.2628	0.2483	0.2551
0.5	–	0.2526	0.2713	0.2551	0.2491	0.2560	0.2594	0.2534	0.2500	0.2483	0.2517	0.2765	0.2449	0.2491	0.2483	–
0.6	–	0.2449	0.2543	0.2594	0.2543	0.2551	0.2526	0.2577	0.2398	0.2619	0.2440	0.2637	0.2517	0.2568	0.2551	0.2449
0.7	–	0.2440	0.2474	0.2551	0.2440	0.2534	0.2526	0.2457	0.2389	0.2568	0.2551	0.2602	0.2483	0.2645	0.2602	0.2517
0.8	–	0.2423	0.2398	0.2585	0.2355	0.2483	0.2577	0.2594	0.2304	0.2662	0.2577	0.2594	0.2474	0.2543	0.2568	0.2483
0.9	–	0.2346	0.2440	0.2398	0.2517	0.2483	0.2585	0.2491	0.2440	0.2432	0.2389	0.2440	0.2526	0.2432	0.2398	0.2363

Table 9: Final ARC-Easy evals with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	0.4672	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.4600	0.4756	0.4760	0.4764	0.4756	0.4785	0.4764	0.4743	0.4689	0.4701	0.4739	0.4714	0.4663	0.4722	0.4853
0.2	–	0.4701	0.4680	0.4840	0.4672	0.4827	0.4705	0.4819	0.4790	0.4832	0.4886	0.4861	0.4811	0.4802	0.4891	0.4773
0.3	–	0.4655	0.4731	0.4781	0.4798	0.4764	0.4903	0.4983	0.4697	0.4916	0.4811	0.4920	0.4697	0.4844	0.4865	0.4903
0.4	–	0.4773	0.4689	0.4739	0.4802	0.4705	0.4815	0.4806	0.4844	0.4907	0.4676	0.4827	0.5000	0.4752	0.4815	0.4823
0.5	–	0.4781	0.4575	0.4769	0.4726	0.4840	0.4836	0.4979	0.4844	0.4899	0.4903	0.4912	0.4899	0.4823	0.4781	–
0.6	–	0.4735	0.4722	0.4714	0.4726	0.4840	0.4865	0.5000	0.4827	0.4832	0.4769	0.4857	0.4954	0.4790	0.4899	0.4836
0.7	–	0.4684	0.4676	0.4705	0.4651	0.4823	0.4811	0.4941	0.4815	0.4886	0.4827	0.4907	0.4764	0.4865	0.4823	0.4756
0.8	–	0.4781	0.4646	0.4600	0.4689	0.4731	0.4735	0.5029	0.4735	0.4962	0.4790	0.4752	0.4697	0.4668	0.4832	0.4840
0.9	–	0.4676	0.4722	0.4562	0.4668	0.4651	0.4668	0.4819	0.4832	0.4827	0.4638	0.4646	0.4781	0.4693	0.4693	0.4621

Table 10: Final BoolQ evals with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	0.5667	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.6049	0.5893	0.5700	0.4606	0.5566	0.5734	0.5832	0.5532	0.6000	0.5208	0.5578	0.5141	0.5951	0.5927	0.5985
0.2	–	0.5911	0.5731	0.5502	0.5878	0.5391	0.5737	0.5982	0.5636	0.5994	0.5703	0.5865	0.5532	0.5456	0.5887	0.5630
0.3	–	0.5734	0.5884	0.5343	0.5575	0.5398	0.5930	0.5609	0.5624	0.5798	0.5752	0.5440	0.5881	0.5391	0.5856	0.5951
0.4	–	0.5379	0.5972	0.5242	0.5483	0.5190	0.5979	0.5538	0.5832	0.5792	0.5560	0.5700	0.5618	0.5765	0.5832	0.5841
0.5	–	0.5489	0.5902	0.5498	0.5780	0.5765	0.5817	0.5440	0.5437	0.5324	0.5618	0.5902	0.5578	0.5615	0.5948	–
0.6	–	0.5462	0.5563	0.5884	0.5642	0.5303	0.5823	0.5679	0.5477	0.5489	0.5349	0.5468	0.5606	0.5862	0.5327	0.5862
0.7	–	0.5853	0.5862	0.5856	0.5761	0.5382	0.5618	0.5752	0.5609	0.5306	0.5474	0.5428	0.5740	0.5520	0.5657	0.5135
0.8	–	0.5850	0.5869	0.5813	0.5807	0.5471	0.5869	0.5336	0.5633	0.5474	0.5682	0.5404	0.5465	0.5945	0.5480	0.5795
0.9	–	0.5789	0.5835	0.5810	0.5523	0.5737	0.5829	0.5924	0.5884	0.5587	0.5725	0.5737	0.5664	0.5826	0.5869	0.5722

Table 11: Final HellaSwag evals with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	0.3634	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.3689	0.3766	0.3861	0.3832	0.3859	0.3865	0.3849	0.3858	0.3875	0.3841	0.3871	0.3853	0.3848	0.3852	0.3846
0.2	–	0.3775	0.3860	0.3905	0.3926	0.3900	0.3909	0.3931	0.3921	0.3881	0.3869	0.3909	0.3891	0.3886	0.3851	0.3900
0.3	–	0.3789	0.3806	0.3897	0.3935	0.3856	0.3899	0.3893	0.3898	0.3907	0.3869	0.3870	0.3902	0.3848	0.3873	0.3858
0.4	–	0.3756	0.3850	0.3914	0.3920	0.3892	0.3911	0.3911	0.3882	0.3872	0.3847	0.3903	0.3860	0.3922	0.3854	0.3857
0.5	–	0.3752	0.3847	0.3889	0.3902	0.3883	0.3923	0.3831	0.3884	0.3854	0.3819	0.3843	0.3826	0.3843	0.3844	–
0.6	–	0.3765	0.3805	0.3876	0.3886	0.3822	0.3809	0.3845	0.3837	0.3807	0.3785	0.3832	0.3816	0.3828	0.3758	0.3787
0.7	–	0.3745	0.3833	0.3865	0.3844	0.3781	0.3774	0.3777	0.3790	0.3773	0.3746	0.3745	0.3713	0.3743	0.3712	0.3731
0.8	–	0.3690	0.3781	0.3818	0.3786	0.3706	0.3704	0.3702	0.3705	0.3687	0.3638	0.3614	0.3675	0.3617	0.3609	0.3593
0.9	–	0.3665	0.3681	0.3676	0.3615	0.3540	0.3575	0.3505	0.3510	0.3439	0.3422	0.3420	0.3347	0.3342	0.3354	0.3360

Table 12: Final OpenBookQA evals with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	0.2920	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.2960	0.2960	0.3160	0.3020	0.3060	0.3060	0.3160	0.3060	0.3000	0.3140	0.2960	0.3080	0.2940	0.3240	0.3120
0.2	–	0.3060	0.3060	0.2980	0.2820	0.2960	0.3160	0.3000	0.2940	0.3040	0.3120	0.3080	0.2920	0.3000	0.3060	0.3040
0.3	–	0.2900	0.3120	0.3000	0.3060	0.2980	0.3080	0.3060	0.3120	0.3160	0.3180	0.3060	0.3100	0.2860	0.3040	0.2940
0.4	–	0.3100	0.2940	0.3060	0.3040	0.3100	0.2900	0.2960	0.2940	0.3140	0.3040	0.2840	0.3020	0.3020	0.3220	0.2920
0.5	–	0.2980	0.3020	0.3160	0.2900	0.3240	0.2860	0.2940	0.2900	0.3180	0.2800	0.2920	0.3060	0.2980	0.3100	–
0.6	–	0.3000	0.2860	0.3180	0.3180	0.3060	0.3040	0.2900	0.2880	0.3000	0.2940	0.2780	0.2880	0.2980	0.3140	0.2940
0.7	–	0.2980	0.2980	0.3100	0.3140	0.3060	0.2800	0.2880	0.2780	0.3060	0.2860	0.2920	0.2900	0.2960	0.2980	0.2900
0.8	–	0.2980	0.2780	0.3100	0.3100	0.3040	0.2860	0.2920	0.2740	0.2940	0.2940	0.2960	0.2940	0.2860	0.2980	0.2980
0.9	–	0.2980	0.2880	0.3100	0.3000	0.3040	0.2760	0.2800	0.2760	0.2860	0.3000	0.2960	0.2820	0.2900	0.3020	0.2880

Table 13: Final PIQA evals with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	0.6638	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.6697	0.6736	0.6719	0.6654	0.6643	0.6714	0.6752	0.6703	0.6817	0.6730	0.6725	0.6757	0.6681	0.6659	0.6736
0.2	–	0.6703	0.6708	0.6741	0.6659	0.6703	0.6714	0.6725	0.6703	0.6681	0.6643	0.6801	0.6659	0.6785	0.6801	0.6692
0.3	–	0.6676	0.6659	0.6665	0.6790	0.6703	0.6708	0.6708	0.6708	0.6708	0.6703	0.6757	0.6708	0.6779	0.6779	0.6616
0.4	–	0.6621	0.6654	0.6643	0.6627	0.6790	0.6719	0.6817	0.6681	0.6763	0.6757	0.6708	0.6746	0.6692	0.6752	0.6665
0.5	–	0.6708	0.6768	0.6578	0.6741	0.6763	0.6665	0.6659	0.6736	0.6736	0.6670	0.6736	0.6730	0.6752	0.6725	–
0.6	–	0.6649	0.6676	0.6605	0.6665	0.6768	0.6714	0.6621	0.6741	0.6714	0.6703	0.6687	0.6681	0.6736	0.6741	0.6741
0.7	–	0.6649	0.6610	0.6632	0.6659	0.6719	0.6572	0.6714	0.6714	0.6752	0.6757	0.6638	0.6610	0.6801	0.6681	0.6567
0.8	–	0.6572	0.6676	0.6545	0.6621	0.6736	0.6654	0.6687	0.6621	0.6654	0.6703	0.6659	0.6578	0.6567	0.6649	0.6627
0.9	–	0.6610	0.6643	0.6513	0.6453	0.6600	0.6643	0.6502	0.6420	0.6534	0.6534	0.6442	0.6436	0.6507	0.6583	0.6534

Table 14: Final Winogrande evals with varying r and s, for 270M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.0	0.5130	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.5193	0.5312	0.5107	0.5130	0.5257	0.5114	0.5193	0.5099	0.5162	0.5375	0.5209	0.5264	0.5091	0.5154	0.5241
0.2	–	0.5114	0.4957	0.5185	0.5146	0.5264	0.5091	0.5391	0.5114	0.5067	0.5185	0.5162	0.5249	0.4988	0.5257	0.5257
0.3	–	0.5367	0.5091	0.5233	0.5233	0.5107	0.5020	0.5083	0.5280	0.5099	0.5280	0.5028	0.4957	0.5225	0.5107	0.5446
0.4	–	0.5130	0.5280	0.5130	0.5233	0.5130	0.5099	0.5241	0.5028	0.5217	0.5099	0.5272	0.5178	0.4925	0.5185	0.5028
0.5	–	0.5414	0.5154	0.5154	0.5138	0.5257	0.5367	0.5233	0.5028	0.5067	0.5178	0.5328	0.5146	0.5154	0.5391	–
0.6	–	0.5146	0.5067	0.5146	0.5162	0.5036	0.5059	0.5296	0.5075	0.5083	0.5162	0.5020	0.5107	0.5264	0.5233	0.5146
0.7	–	0.5170	0.5099	0.4988	0.5004	0.5178	0.5083	0.5059	0.5217	0.5043	0.5091	0.5280	0.5004	0.4988	0.5351	0.5051
0.8	–	0.5233	0.5051	0.4901	0.5114	0.5028	0.5178	0.5185	0.5146	0.5178	0.5170	0.5075	0.5091	0.5099	0.5012	0.5114
0.9	–	0.5209	0.5020	0.5028	0.5020	0.4972	0.5067	0.5130	0.5162	0.5146	0.5114	0.5114	0.4925	0.5249	0.5020	0.5020

Table 15: Final ARC-Challenge evals with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.2619	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.2568	0.2619	0.2534	0.2671	0.2654	0.2654	0.2654	0.2688	0.2645	0.2440	0.2543	0.2782
0.2	–	0.2517	0.2654	0.2662	0.2637	0.2645	0.2577	0.2619	0.2628	0.2773	0.2594	0.2730	0.2756
0.3	–	0.2611	0.2577	0.2705	0.2568	0.2551	0.2534	0.2696	0.2602	0.2662	0.2739	0.2688	0.2645
0.4	–	0.2628	0.2568	0.2517	0.2568	0.2611	0.2722	0.2739	0.2662	0.2654	0.2585	0.2611	0.2611
0.5	–	0.2449	0.2585	0.2577	0.2662	0.2645	0.2628	0.2602	0.2637	0.2602	0.2534	0.2628	0.2688
0.6	–	0.2500	0.2602	0.2543	0.2585	0.2577	0.2577	0.2611	0.2602	0.2773	0.2739	0.2611	0.2560
0.7	–	0.2534	0.2705	0.2637	0.2560	0.2662	0.2705	0.2637	0.2645	0.2628	0.2705	0.2713	0.2679
0.8	–	0.2432	0.2611	0.2619	0.2432	0.2628	0.2594	0.2645	0.2483	0.2662	0.2619	0.2722	0.2602
0.9	–	0.2551	0.2602	0.2747	0.2483	0.2517	0.2474	0.2491	0.2747	0.2585	0.2466	0.2611	0.2594

Table 16: Final ARC-Easy evals with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.4747	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.4870	0.4941	0.4903	0.5029	0.5202	0.4954	0.5004	0.5198	0.5114	0.4992	0.4920	0.5088
0.2	–	0.4731	0.4924	0.4907	0.5332	0.5008	0.4895	0.4983	0.5130	0.4992	0.5051	0.5223	0.5034
0.3	–	0.4815	0.4979	0.5021	0.5160	0.5034	0.5181	0.5278	0.5173	0.5067	0.4945	0.5156	0.5114
0.4	–	0.4794	0.5025	0.4899	0.5051	0.5072	0.5017	0.5303	0.5223	0.5236	0.5181	0.5147	0.5021
0.5	–	0.4693	0.5038	0.5013	0.5189	0.4992	0.5156	0.5114	0.5072	0.5029	0.5177	0.5046	0.4983
0.6	–	0.4769	0.5093	0.5093	0.4920	0.5076	0.4962	0.5004	0.5059	0.4979	0.5004	0.5101	0.5114
0.7	–	0.4735	0.4933	0.4992	0.4832	0.4954	0.4987	0.4895	0.5000	0.5114	0.5122	0.4979	0.4827
0.8	–	0.4663	0.4836	0.4975	0.4886	0.4920	0.5067	0.5101	0.5042	0.4962	0.5059	0.5105	0.4987
0.9	–	0.4760	0.4920	0.4907	0.4823	0.4823	0.5021	0.5076	0.4996	0.4794	0.4815	0.5059	0.4853

Table 17: Final BoolQ evals with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.5847	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.5223	0.5606	0.5245	0.5713	0.5618	0.5076	0.5933	0.5572	0.5621	0.5336	0.5550	0.5786
0.2	–	0.5260	0.6000	0.5318	0.5960	0.5431	0.5798	0.5462	0.5621	0.5083	0.5713	0.5746	0.5309
0.3	–	0.5391	0.5841	0.5535	0.5410	0.5789	0.5761	0.5566	0.5847	0.4927	0.5352	0.5190	0.5673
0.4	–	0.5131	0.5713	0.5575	0.5557	0.5187	0.5685	0.5557	0.5336	0.5382	0.5272	0.4838	0.5872
0.5	–	0.6000	0.5046	0.5612	0.5468	0.5700	0.5596	0.5676	0.5930	0.5554	0.5685	0.5468	0.5789
0.6	–	0.6141	0.5092	0.5083	0.5979	0.5957	0.5988	0.5416	0.5300	0.5297	0.5339	0.5367	0.5633
0.7	–	0.5966	0.5306	0.5113	0.5480	0.5899	0.5471	0.5694	0.5346	0.5080	0.5119	0.4957	0.5318
0.8	–	0.5869	0.5495	0.4936	0.5615	0.5511	0.5859	0.5544	0.5621	0.4966	0.4547	0.5220	0.5554
0.9	–	0.6162	0.5431	0.5024	0.5832	0.5388	0.5719	0.5758	0.5709	0.5116	0.5554	0.5462	0.5560

Table 18: Final HellaSwag evals with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.4021	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.4107	0.4190	0.4237	0.4225	0.4252	0.4284	0.4281	0.4261	0.4276	0.4227	0.4292	0.4309
0.2	–	0.4122	0.4176	0.4260	0.4277	0.4266	0.4269	0.4267	0.4267	0.4262	0.4288	0.4243	0.4280
0.3	–	0.4143	0.4179	0.4306	0.4275	0.4261	0.4260	0.4265	0.4284	0.4238	0.4275	0.4243	0.4236
0.4	–	0.4137	0.4200	0.4265	0.4340	0.4279	0.4248	0.4290	0.4266	0.4261	0.4291	0.4246	0.4247
0.5	–	0.4145	0.4227	0.4247	0.4272	0.4246	0.4282	0.4219	0.4251	0.4219	0.4256	0.4219	0.4266
0.6	–	0.4144	0.4192	0.4226	0.4250	0.4244	0.4231	0.4256	0.4262	0.4200	0.4196	0.4180	0.4180
0.7	–	0.4124	0.4192	0.4214	0.4229	0.4195	0.4206	0.4188	0.4220	0.4141	0.4164	0.4160	0.4148
0.8	–	0.4102	0.4164	0.4224	0.4152	0.4153	0.4171	0.4175	0.4132	0.4120	0.4113	0.4068	0.4077
0.9	–	0.4100	0.4091	0.4089	0.4097	0.4019	0.4020	0.4049	0.3976	0.3950	0.3940	0.3921	0.3919

Table 19: Final OpenBookQA evals with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.3060	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.3200	0.3020	0.3120	0.3100	0.3180	0.3080	0.2980	0.3140	0.3140	0.3140	0.3200	0.3340
0.2	–	0.3040	0.3220	0.3180	0.3220	0.3220	0.3020	0.3040	0.3100	0.3180	0.3220	0.3040	0.3240
0.3	–	0.3260	0.3400	0.3120	0.3260	0.3240	0.3220	0.3220	0.3180	0.3300	0.3060	0.3160	0.3160
0.4	–	0.2980	0.3100	0.3000	0.3220	0.3100	0.3080	0.3060	0.3200	0.3280	0.3040	0.3080	0.3040
0.5	–	0.3200	0.3160	0.3180	0.3240	0.3060	0.3220	0.3180	0.3000	0.3200	0.3140	0.3160	0.3260
0.6	–	0.3300	0.3300	0.3240	0.3040	0.3080	0.3120	0.3180	0.2980	0.3060	0.3080	0.3160	0.3240
0.7	–	0.2980	0.3080	0.3080	0.3180	0.3100	0.2900	0.3040	0.3060	0.2940	0.3080	0.3340	0.3020
0.8	–	0.3060	0.3160	0.3080	0.3220	0.3060	0.3120	0.2980	0.3200	0.2860	0.3320	0.2920	0.3020
0.9	–	0.2860	0.3280	0.3100	0.3180	0.3200	0.3060	0.3120	0.3140	0.2980	0.3080	0.2960	0.3020

Table 20: Final PIQA evals with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.6730	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.6746	0.6844	0.6785	0.6785	0.6942	0.6855	0.6861	0.6774	0.6834	0.6861	0.6812	0.6904
0.2	–	0.6812	0.6779	0.6801	0.6779	0.6921	0.6861	0.6899	0.6850	0.6844	0.6844	0.6861	0.6801
0.3	–	0.6785	0.6817	0.6844	0.6801	0.6904	0.6752	0.6790	0.6855	0.6823	0.6877	0.6893	0.6910
0.4	–	0.6790	0.6839	0.6828	0.6823	0.6872	0.6779	0.6839	0.6937	0.6823	0.6964	0.6866	0.6861
0.5	–	0.6844	0.6779	0.6785	0.6801	0.6937	0.6855	0.6855	0.6921	0.6844	0.6915	0.6877	0.6899
0.6	–	0.6806	0.6834	0.6834	0.6882	0.6899	0.6834	0.6948	0.6888	0.6834	0.6823	0.6839	0.6828
0.7	–	0.6708	0.6752	0.6757	0.6779	0.6926	0.6763	0.6921	0.6866	0.6823	0.6806	0.6806	0.6741
0.8	–	0.6665	0.6752	0.6779	0.6817	0.6844	0.6855	0.6806	0.6823	0.6844	0.6768	0.6828	0.6795
0.9	–	0.6665	0.6654	0.6692	0.6844	0.6806	0.6779	0.6861	0.6757	0.6741	0.6714	0.6681	0.6659

Table 21: Final Winogrande evals with varying r and s, for 270M models trained for 100k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.5154	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.5170	0.5351	0.5036	0.5209	0.5343	0.5272	0.5272	0.5351	0.5328	0.5391	0.5446	0.5122
0.2	–	0.5193	0.5288	0.5304	0.5406	0.5454	0.5422	0.5272	0.5399	0.5288	0.5383	0.5193	0.5343
0.3	–	0.5185	0.5170	0.5320	0.5257	0.5422	0.5296	0.5414	0.5193	0.5304	0.5296	0.5375	0.5288
0.4	–	0.5288	0.5280	0.5201	0.5414	0.5257	0.5406	0.5383	0.5470	0.5201	0.5288	0.5375	0.5280
0.5	–	0.5075	0.5146	0.5335	0.5304	0.5320	0.5430	0.5320	0.5304	0.5162	0.5343	0.5264	0.5438
0.6	–	0.4957	0.5296	0.5241	0.5312	0.5383	0.5296	0.5375	0.5383	0.5288	0.5091	0.5462	0.5414
0.7	–	0.5130	0.5225	0.5375	0.5249	0.5264	0.5320	0.5391	0.5264	0.5225	0.5280	0.5296	0.5209
0.8	–	0.5233	0.5257	0.5083	0.5051	0.5430	0.5272	0.5272	0.5257	0.5304	0.5280	0.5359	0.5114
0.9	–	0.5280	0.5178	0.5170	0.5193	0.5178	0.5170	0.5083	0.5178	0.5075	0.5209	0.5067	0.5178

Table 22: Final ARC-Challenge evals with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.2551	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.2628	0.2671	0.2816	0.2747	0.2833	0.2952	0.2739	0.2892	0.2850	0.2765	0.2833	0.2790
0.2	–	0.2679	0.2765	0.2782	0.2782	0.2790	0.2867	0.3029	0.2867	0.2799	0.2918	0.3012	0.2927
0.3	–	0.2645	0.2765	0.2688	0.2705	0.2688	0.2944	0.2824	0.2824	0.2816	0.2961	0.2969	0.2816
0.4	–	0.2628	0.2782	0.2705	0.2850	0.2730	0.2867	0.2875	0.2910	0.2901	0.2850	0.2892	0.3063
0.5	–	0.2765	0.2807	0.2884	0.2782	0.2611	0.2961	0.2824	0.2910	0.2961	0.2807	0.3012	0.2952
0.6	–	0.2688	0.2696	0.2799	0.2773	0.2671	0.2910	0.2765	0.2858	0.2935	0.2824	0.2995	0.2892
0.7	–	0.2696	0.2688	0.2867	0.2688	0.2782	0.2816	0.2858	0.2816	0.2875	0.2961	0.2910	0.2918
0.8	–	0.2747	0.2875	0.2944	0.2799	0.2696	0.2637	0.2824	0.2884	0.2875	0.2867	0.2816	0.2782
0.9	–	0.2645	0.2722	0.2671	0.2816	0.2688	0.2628	0.2594	0.2705	0.2705	0.2773	0.2747	0.2730

Table 23: Final ARC-Easy evals with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.5168	-	-	-	-	-	-	-	-	-	-	-	-
0.1	-	0.5210	0.5530	0.5412	0.5307	0.5320	0.5286	0.5391	0.5219	0.5383	0.5370	0.5480	0.5526
0.2	-	0.5261	0.5328	0.5543	0.5429	0.5202	0.5362	0.5442	0.5501	0.5450	0.5644	0.5564	0.5593
0.3	-	0.5152	0.5391	0.5450	0.5261	0.5248	0.5383	0.5320	0.5543	0.5535	0.5492	0.5606	0.5455
0.4	-	0.5173	0.5438	0.5610	0.5265	0.5236	0.5379	0.5467	0.5400	0.5522	0.5467	0.5619	0.5556
0.5	-	0.5118	0.5349	0.5484	0.5362	0.5248	0.5442	0.5299	0.5547	0.5450	0.5425	0.5661	0.5593
0.6	-	0.5038	0.5459	0.5412	0.5455	0.5417	0.5467	0.5417	0.5543	0.5665	0.5467	0.5657	0.5518
0.7	-	0.5093	0.5366	0.5396	0.5425	0.5379	0.5539	0.5311	0.5614	0.5459	0.5421	0.5530	0.5366
0.8	-	0.5093	0.5400	0.5446	0.5375	0.5332	0.5463	0.5370	0.5341	0.5442	0.5497	0.5450	0.5332
0.9	-	0.5000	0.5370	0.5311	0.5236	0.5311	0.5290	0.5215	0.5223	0.5404	0.5265	0.5408	0.5434

Table 24: Final BoolQ evals with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.5661	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.5924	0.5924	0.5896	0.6098	0.5517	0.5920	0.6214	0.6031	0.5722	0.5697	0.5642	0.5789
0.2	–	0.5761	0.5991	0.5994	0.6254	0.6070	0.5783	0.5612	0.5364	0.5300	0.5327	0.6110	0.6012
0.3	–	0.6000	0.5810	0.5988	0.6024	0.5477	0.5336	0.5333	0.5483	0.5462	0.5456	0.5865	0.5670
0.4	–	0.5661	0.5872	0.5963	0.6037	0.5703	0.5297	0.5664	0.5703	0.5550	0.5532	0.5768	0.5911
0.5	–	0.5832	0.6009	0.6052	0.6162	0.5615	0.5697	0.5471	0.5070	0.5673	0.5119	0.5330	0.5985
0.6	–	0.5394	0.5823	0.6257	0.5942	0.5899	0.6024	0.5520	0.5654	0.5410	0.5346	0.5453	0.5664
0.7	–	0.5798	0.5627	0.6110	0.6043	0.5847	0.5624	0.5780	0.5777	0.5587	0.5125	0.6135	0.5884
0.8	–	0.5813	0.5838	0.6187	0.6168	0.5960	0.5758	0.5636	0.5847	0.5453	0.5211	0.5905	0.5755
0.9	–	0.5761	0.5869	0.5960	0.6183	0.5645	0.5355	0.5498	0.5905	0.5829	0.5951	0.6110	0.5648

Table 25: Final HellaSwag evals with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.4347	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.4554	0.4658	0.4780	0.4763	0.4736	0.4760	0.4756	0.4750	0.4771	0.4739	0.4787	0.4764
0.2	–	0.4628	0.4711	0.4806	0.4802	0.4810	0.4823	0.4855	0.4832	0.4813	0.4797	0.4809	0.4811
0.3	–	0.4620	0.4726	0.4788	0.4832	0.4817	0.4777	0.4840	0.4837	0.4825	0.4825	0.4793	0.4806
0.4	–	0.4613	0.4763	0.4797	0.4814	0.4802	0.4804	0.4812	0.4811	0.4792	0.4727	0.4749	0.4753
0.5	–	0.4642	0.4783	0.4812	0.4808	0.4781	0.4795	0.4767	0.4795	0.4757	0.4810	0.4718	0.4725
0.6	–	0.4614	0.4710	0.4780	0.4754	0.4755	0.4717	0.4695	0.4721	0.4706	0.4689	0.4671	0.4631
0.7	–	0.4588	0.4682	0.4740	0.4702	0.4706	0.4648	0.4679	0.4654	0.4590	0.4593	0.4556	0.4527
0.8	–	0.4561	0.4658	0.4685	0.4588	0.4582	0.4568	0.4524	0.4485	0.4465	0.4448	0.4414	0.4367
0.9	–	0.4494	0.4507	0.4482	0.4417	0.4343	0.4265	0.4257	0.4207	0.4107	0.4076	0.4033	0.4005

Table 26: Final OpenBookQA evals with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.3120	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.3200	0.3220	0.3280	0.3440	0.3460	0.3320	0.3480	0.3160	0.3540	0.3560	0.3220	0.3480
0.2	–	0.3360	0.3260	0.3320	0.3380	0.3160	0.3300	0.3220	0.3280	0.3360	0.3260	0.3340	0.3380
0.3	–	0.3280	0.3340	0.3280	0.3100	0.3500	0.3340	0.3480	0.3340	0.3420	0.3240	0.3400	0.3180
0.4	–	0.3320	0.3340	0.3440	0.3320	0.3460	0.3140	0.3220	0.3340	0.3300	0.3400	0.3380	0.3420
0.5	–	0.3320	0.3420	0.3180	0.3300	0.3280	0.3260	0.3360	0.3420	0.3060	0.3220	0.3220	0.3200
0.6	–	0.3320	0.3340	0.3320	0.3200	0.3320	0.3260	0.3240	0.3500	0.3200	0.3260	0.3460	0.3320
0.7	–	0.3380	0.3380	0.3480	0.3000	0.3380	0.3300	0.3500	0.3500	0.3140	0.3320	0.3400	0.3240
0.8	–	0.3260	0.3340	0.3240	0.3180	0.3160	0.3200	0.3400	0.3480	0.3380	0.3380	0.3240	0.3200
0.9	–	0.3040	0.3280	0.3140	0.3060	0.3080	0.3160	0.3280	0.3440	0.3180	0.3340	0.3060	0.3140

Table 27: Final PIQA evals with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.6904	-	-	-	-	-	-	-	-	-	-	-	-
0.1	-	0.6991	0.7062	0.7002	0.7024	0.6997	0.7155	0.7073	0.7095	0.7008	0.7057	0.7062	0.6964
0.2	-	0.7057	0.7089	0.6964	0.7067	0.7029	0.7127	0.6980	0.7095	0.7127	0.7089	0.7029	0.7089
0.3	-	0.7018	0.7095	0.7013	0.7024	0.7057	0.7127	0.7051	0.7144	0.7062	0.7051	0.7095	0.7127
0.4	-	0.7024	0.7067	0.7008	0.7095	0.6980	0.7013	0.7067	0.7138	0.7198	0.7100	0.7127	0.7078
0.5	-	0.6997	0.7057	0.7122	0.7084	0.6997	0.7040	0.7111	0.7127	0.7100	0.7002	0.7078	0.7008
0.6	-	0.6937	0.6986	0.7073	0.6970	0.7046	0.7067	0.7067	0.7127	0.7160	0.7084	0.7029	0.6915
0.7	-	0.6980	0.7062	0.7013	0.6953	0.7084	0.7029	0.7008	0.7160	0.7024	0.7057	0.7002	0.6959
0.8	-	0.7057	0.6921	0.7018	0.7024	0.6980	0.6991	0.7008	0.7073	0.7062	0.7018	0.7024	0.6991
0.9	-	0.6877	0.6975	0.6942	0.6866	0.6877	0.6834	0.6861	0.6948	0.6910	0.6882	0.6882	0.6757

Table 28: Final Winogrande evals with varying r and s, for 600M models trained for 20k total steps. See Table 1 for training details.

r \ s	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.5257	–	–	–	–	–	–	–	–	–	–	–	–
0.1	–	0.5359	0.5446	0.5493	0.5438	0.5493	0.5533	0.5335	0.5478	0.5517	0.5604	0.5430	0.5422
0.2	–	0.5643	0.5280	0.5493	0.5675	0.5462	0.5478	0.5406	0.5359	0.5564	0.5485	0.5517	0.5399
0.3	–	0.5328	0.5320	0.5430	0.5509	0.5391	0.5620	0.5328	0.5320	0.5454	0.5501	0.5454	0.5596
0.4	–	0.5288	0.5422	0.5193	0.5509	0.5501	0.5667	0.5320	0.5383	0.5375	0.5422	0.5438	0.5422
0.5	–	0.5328	0.5493	0.5375	0.5375	0.5517	0.5604	0.5391	0.5501	0.5446	0.5399	0.5264	0.5754
0.6	–	0.5509	0.5391	0.5249	0.5501	0.5470	0.5509	0.5304	0.5454	0.5343	0.5580	0.5406	0.5596
0.7	–	0.5517	0.5549	0.5335	0.5438	0.5478	0.5509	0.5391	0.5335	0.5264	0.5406	0.5280	0.5422
0.8	–	0.5335	0.5399	0.5501	0.5359	0.5525	0.5383	0.5328	0.5304	0.5162	0.5296	0.5178	0.5241
0.9	–	0.5335	0.5414	0.5485	0.5335	0.5509	0.5406	0.5154	0.5296	0.5043	0.5162	0.5241	0.5107