Scaled Signed Averaging Improves In-Context and Early Learning Benchmark Performance in Small Transformers

Anonymous ACL submission

Abstract

While Large Language models' abilities for in-context learning (ICL) have had much success, they have limitations on simple semantic tasks involving quantifiers like *every* and *some*, as well as on tasks with linear functions. We analyze those limitations and identify Softmax, the scoring function in the attention mechanism, as a contributing factor to these limitations. Our **scaled signed averaging (SSA)**, a novel scoring function mitigates these limitations. SSA significantly improves performance on our ICL tasks. In addition, SSA outperforms transformer models with Softmax on several early learning NLP benchmarks and linguistic probing tasks on zero and few-shot settings.

1 Introduction

Scoring functions are a core component of the attention mechanism of transformer architectures, governing how information is aggregated across tokens and crucial for in-context learning (ICL). Despite the empirical success of ICL, however, recent studies have identified systematic limitations in its generalization behavior (McCoy et al., 2024; Ye et al., 2024). Large language models tend to perform well in contexts that resemble patterns frequently encountered during training, but struggle in so-called *low-probability* settings—tasks, data distributions, or algorithmic structures that are rare or absent from the training corpus.

We investigate the origins of these limitations in a controlled setting. We study two simple ICL tasks using transformer models trained from scratch, allowing us to isolate architectural effects from pretraining artifacts. We show that the use of the Softmax scoring function in attention is one contributor to the observed generalization failures.

To improve generalization, we introduce *scaled signed averaging* (SSA), a trainable alternative to Softmax for attention scoring. We show that SSA substantially improves ICL generalization in small Transformer models in our controlled tasks. We further demonstrate that these gains extend beyond synthetic settings: a decoder-only GPT-2–style model trained from scratch on the *FineWebText* corpus (Penedo et al., 2024) with SSA achieves lower perplexity and stronger performance than a Softmax-based counterpart across several standard NLP benchmarks for early-stage Transformer evaluation, under both zero-shot and few-shot conditions. We also compare encoder models using SSA to their Softmax-based counterparts. In particular, we train several versions of BabyBERTa on a corpus of child-directed speech, including a standard Softmax model and SSA variants. Consistent with our decoder-only results, the SSA-based BabyBERTa models achieve superior performance on many tasks in the grammatical probing suite of (Huebner et al., 2021).

2 Related Work

In-Context Learning (Brown et al., 2020) introduced ICL as a paradigm in which a model learns at inference time from the prompt by analogy, without modifying any training parameters. (Dong et al., 2022) survey the successes and challenges of ICL, noting that existing research has primarily focused "on simple tasks and small models", such as learning linear or basic Boolean functions. A reason for this focus is that ICL capacity emerges through training from scratch, so studying it requires extensive training. We investigate here versions of both tasks.

Function Learning and Quantification Transformers trained from scratch can perform ICL of simple functions under favorable conditions. (Garg et al., 2022) demonstrated successful ICL of linear functions when training and test distributions match[1], and (Bhattamishra et al., 2023) showed

[1]This work, like ours, follows a general approach to ICL and learning: a model has learned a set $S \subset V$ with predic-

ICL of Boolean functions in small GPT-2–style models. (Raventós et al., 2024) analyzed how ICL capabilities evolve with pretraining size under matched distributions. We evaluate ICL using both linear function prediction and quantification. For quantification, (Asher et al., 2023) proposed assessing a generative model's understanding by encoding semantic situations in input sequences. We adopt their approach, testing a model's grasp of basic quantification concepts by encoding situations in the context and evaluating its ability to interpret them correctly.

Limits of In-Context Learning A growing body of literature highlights sharp limits of ICL under distribution shift. Performance degrades when inference-time inputs differ from the training data (Xie et al., 2021; Zhang et al., 2024; Giannou et al., 2024). (Naim and Asher, 2024) systematically studied distribution shifts that result in substantial degradation, in contrast to (Garg et al., 2022), who only considered small perturbations. (Ye et al., 2024; McCoy et al., 2024) demonstrate general limits of autoregressive training. (Naim and Asher, 2024) describe how model behavior degrades and show that the lack of generalizability is neither due to the models overfitting the data nor a result of memorization.

Problems with Softmax (Tian et al., 2023; Vig, 2019; Htut et al., 2019) show that the softmax operation in attention has concentration effects: attention weights often collapse onto a small set of co-occurring tokens, neglecting others; and in early layers, many heads attend almost exclusively to the first token. Models may rely on only a few tokens, or even a single token per prompt, for prediction (Sekhsaria et al., 2025), with the decision weight concentrated on the final token (Mamidanna et al., 2025). Alternative normalization functions attempt to address these limitations. Sparsemax (Martins and Astudillo, 2016) replaces Softmax with a projection onto the probability simplex, yielding sparse attention distributions that can zero out irrelevant tokens. Entmax (Peters et al., 2019) generalizes both Softmax and Sparsemax through a parametric family, enabling controllable sparsity between dense and sparse attention distributions. Temperature-scaled Softmax (Hinton et al., 2015) adjusts the entropy of the attention distribution via a temperature parameter, while more recent methods such as SA-Softmax (Zheng et al., 2025) and CosFormer (Qin et al., 2022) explore architectural alternatives to Softmax-based attention.

NLP benchmarks (Huebner et al., 2021) demonstrate that transformer-based masked language models can effectively learn core grammatical structures from a small, child-directed corpus. their BabyBERTa achieves a grammatical understanding comparable to RoBERTa-base pre-trained on 30B words. (Huebner et al., 2021) also develop a grammar evaluation suite tailored to child-level vocabularies. We use this suite to compare SSA and softmax as scoring functions in encoder-only models.

3 Our ICL tasks and experimental set up

We study scoring functions in ICL with respect to two tasks. The first is a semantic task about the meaning of the quantifiers *"every"* and *"some"* in a restricted setting; the model is given examples of sequences of numbers with a certain property P and is prompted to predict whether every element or some element in the sequence satisfies P. The second task is a mathematical task in which the model learns linear functions in context. Both tasks use clean, simple data (sequences of numbers) and transparent prompting, which helps mitigate issues related to prompt engineering. Since ICL capabilities depend on pre-training, we train and study models trained from scratch in well-controlled settings to analyze ICL limits. Our training reflects the standard NLP next token prediction training of language models; but our data and tasks are simpler than with most NLP tasks. We can thus analyze specific components of the transformer and study the limitations that motivate SSA's design.

Our quantification task[2] is to predict the truth of the simple quantified sentences (1-a) and (1-b) given a contextually given string of numbers of length 40. Our training set of strings S contain numbers chosen from a training distribution $D_{\mathcal{I}}$, which we set to the Gaussian distribution $\mathcal{N}(0, 1)$.

(1) a. Every number in the sequence is positive.
b. Some number in the sequence is positive.

At inference time, we test generalization by shifting

tion $\hat{S}$ if, when sampling from V at test time using the same distribution as during training, the expected error for $x \in V$ with $\neg(x \in \hat{S} \leftrightarrow x \in S)$ is close to 0.

[2]See Figure 2 for an example of the task

both the input distribution $D_{\mathcal{I}}^{\text{test}}$ and the sequence length S^{test}. The set S^{test} includes sequences ranging in length from 10 to 200, while $D_{\mathcal{I}}^{\text{test}}$ ranges over Gaussian distributions $\mathcal{N}(0, \sigma)$ for $\sigma \geq 1$.

In the linear function task, the target function is an affine map of the form $y_i = f(x_i) = ax_i + b$. To construct the training set, we first sample coefficients a and b for f from a distribution denoted $D_{\mathcal{F}}$. Each training sequence is then populated with input elements x_i drawn from a separate distribution $D_{\mathcal{I}}$. At inference time, we evaluate generalization by shifting both the input distribution $D_{\mathcal{I}}^{\text{test}} \sim \mathcal{N}(0, \sigma_1)$ for $\sigma_1 \geq 1$ and the function parameters distribution $D_{\mathcal{F}}^{\text{test}} \sim \mathcal{N}(0, \sigma_2)$ for $\sigma_2 \geq 1$.

Despite their simplicity, our tasks are conceptually fundamental. Failure on the quantification task suggests deep limitations in a model's reasoning capabilities. The notion of logical or semantic consequence, for instance, involves quantification; a failure to understand quantification implies a failure to understand what it means to reason correctly and will entail mistakes in tasks like question answering (Chaturvedi et al., 2024).

The function prediction task tests a model's ability to extrapolate patterns from contextual data to novel situations, a core requirement of ICL. While large models often succeed by relying on extensive encoded knowledge, true generalization requires the ability to go beyond the training distribution. Our function task isolates this challenge in a controlled setting. If a model fails to generalize here, we should be cautious about claims of generalization in more complex, less controlled scenarios involving noisy or unknown data.

Our ICL tasks involve training from scratch on sequences that contain in-context examples (input-output pairs) of the form $(x_1, y_1, ..., x_i)$ ending with a query input x_i, for which the model must predict the corresponding output y_i. Inputs are sampled from one or more training distributions. We employ curriculum learning on a set S of training sequences of varying lengths, ranging from 1 to 40.

We trained several transformer models[3] from scratch, ranging from small 1 layer model to models with 18 layers, 8 attention heads, and an embedding size of 256. We feature results here from a 12 layers and 8 attention heads (22.5M parameters) model with and without MLP on our tasks, as larger models did not yield significantly better predictions. To identify the component responsible for ICL in transformers and what hinders their generalization, we did an ablation study by removing components from the architecture to examine ICL tasks in models with fewer components.

We train a model $\mathcal{L}_\theta$ parameterized by θ to minimize the expected loss over all prompts:

$$\min_{\theta} \mathbb{E}_P \left[\frac{1}{k+1} \sum_{i=0}^{k} \ell \left(y_{i+1}, \mathcal{L}_\theta \left((x_1, y_1, ..., x_{i+1}) \right) \right) \right] \quad (1)$$

where $\ell(.,.)$ represents the loss function: we use squared error for the linear function task and cross-entropy for the quantifier task. In the quantifier task, y represents the ground truth given a sequence ending with the input x. In the function task, y is the ground truth value of $f(x)$, where f is the underlying function generating the sequence up to x. We train models for 500,000 steps using a batch size of 64.

For the quantifier task, we evaluate a model's ICL performance on each pair $(S^{\text{test}}, D_{\mathcal{I}}^{\text{test}})$ by generating 100 samples, each consisting of 64 batches. For each sample, the model receives a score of 1 if the model's prediction is incorrect and 0 otherwise. The final evaluation measure is obtained by averaging the error across all samples.

For the linear function task, we assess ICL performance on each pair $(D_{\mathcal{I}}^{\text{test}}, D_{\mathcal{F}}^{\text{test}})$ by sampling $N = 100$ functions from $D_{\mathcal{F}}^{\text{test}}$. For each function f_j, we generate $N_b = 64$ batches, each containing $N_p = 40$ input points drawn from $D_{\mathcal{I}}^{\text{test}}$. Within each batch b, we predict $f_j(x_k^b)$ for every x_k^b with $k \geq 2$, given the prompt sequence $(x_1^b, f_j(x_1^b), ..., x_{k-1}^b, f_j(x_{k-1}^b), x_k^b)$. For each function, we compute the mean squared error (MSE) across all predicted points in all batches. The final ICL performance metric is obtained by averaging the MSE across all sampled functions:

$$\epsilon_\sigma = \frac{1}{N} \sum_{j=1}^{N} \sum_{b=1}^{N_b} \frac{1}{N_b} \left(\frac{1}{N_p} \sum_{k=3}^{N_p} (\text{pred}_k^{b,j} - f_j(x_k^b))^2 \right) \quad (2)$$

This evaluation (2) across different distributions provides a comprehensive assessment of the model's ability to generalize its learning.

4 ICL for quantifiers and linear functions

ICL with quantifiers Experiments on quantification task showed that when test samples are drawn from the same distribution as training, i.e.

[3]Our models were Decoder-only (GPT-2). Our code can be found in https://anonymous.4open.science/r/SSA/

$D_{\mathcal{I}}, D_{\mathcal{I}}^{\text{test}} \sim \mathcal{N}(0,1)$, models successfully learned to predict the correct truth values for (1-a) or (1-b) on test sequences S^{test} that were significantly longer than those seen during training S, as illustrated in Figure 4. However, model performance dropped sharply when inference inputs included one or more x_i values far outside the training distribution (see Figure 2). We refer to such a sequence as *deviant*.

ICL with linear functions We replicated the findings of (Naim and Asher, 2024) in the linear function task: when both training and test data were sampled from $\mathcal{N}(0,1)$, even small models achieved near-zero average error; but all models had systematic non 0 average errors when the target function f was sampled from a shifted distribution $D_{\mathcal{F}}^{\text{test}} = \mathcal{N}(0,\sigma)$ for $\sigma > 2$ (Appendix M).

5 Error analysis

In the quantification task, we found that models base their predictions for an entire deviant sequence S solely on their prediction for the largest element in S (see Figure 2). The presence of a single sufficiently large number in S was enough to trigger this behavior consistently. In the linear function task, we also observed (Naim and Asher, 2024)'s *boundary values* —values that the model fails to exceed during inference (see Figures 1 and 9). These boundary values are responsible for generalization failures: they restrict the model to generate outputs only within a specific range, effectively preventing the model from generalizing its good performance on the task over a small interval to values outside that interval. We found such boundary effects in both attention-only and full transformer models across all our training and testing setups.

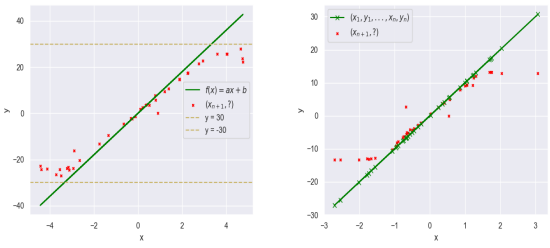

Figure 1: Plots showing examples of boundary values for different models. (Left) Full transformer 12L8AH model tested on $f(x) = 9x$ and (Right) Transformer 12L8AH without MLP model trained on $D_{\mathcal{I}} = D_{\mathcal{F}} = \mathcal{N}(0,1)$ and tested on $f(x) = 10x$.

5.1 Comparison with larger fine-tuned and prompted LLMs

We observed similar issues with larger pre-trained models. We evaluated performance on the quantification task using both fine-tuned (see Appendix A for details) and prompted versions of LLaMA 3.1 8B, as well as the prompted LLaMA 3.3 70B model[4]. In a 5-shot setting, prompted LLaMA 3.1 8B failed to master numerical inputs from $D_{\mathcal{I}}^{\text{test}}$ and showed no generalization to longer sequences. LLaMA 3.3 70B performed better on numerical inputs drawn from distributions outside $\mathcal{N}(0,1)$ but, similarly, failed to generalize to longer sequence lengths. Interestingly, the fine-tuned LLaMA 3.1 8B was able to handle large numbers within a sequence, as shown in Figure 7, but still did not generalize beyond sequence lengths seen in training. We also tested the prompted LLaMA 3.3 70B on the linear function task with inputs and target functions sampled from $D_{\mathcal{I}}, D_{\mathcal{F}} \sim \mathcal{N}(0,1)$. While the model sometimes appeared to assume linearity and apply a regression-like strategy, it still underperformed relative to our small models (see Table 1).

5.2 Ablation studies: the sources of ICL and limits to generalization

Generalization from training data on our ICL tasks challenged not only our smaller models but also much larger ones. This suggests that the problems we identify arise from architectural limitations rather than model scale. We looked at what might be responsible for ICL and its limitations. As with (Olsson et al., 2022), we found that ICL was effective on our tasks even in models composed solely of attention layers, with no feedforward components (FF); these attention-only models performed comparably to their full transformer counterparts (see Figure 11). In contrast, small models consisting only of FF layers failed to perform ICL. This indicates that the attention mechanism is both necessary and sufficient for ICL on our tasks. As in (Naim and Asher, 2024), models without FF components also show boundary values, which means that boundary values originate from the multi-head attention itself.

We then examined various components of our models to identify the source of their generalization failures. To understand why the models struggled to generalize on the quantification task, we first tested whether they could correctly classify indi-

[4]Prompts are provided in the appendix K

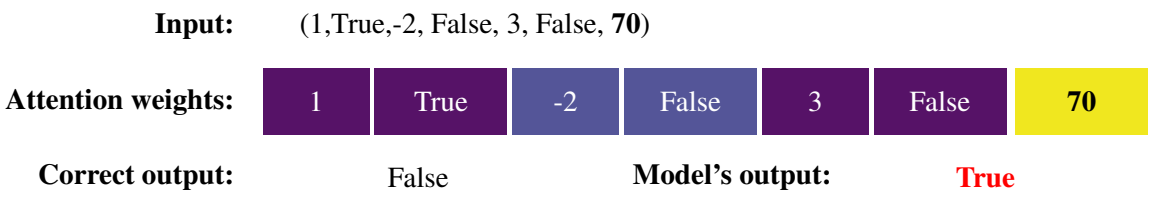

Figure 2: Attention maps for an ICL example for the task "every" of type $(x_1, f(x_1), x_2, f(x_2), ..., x_n)$, where the query x_n is a big value.

vidual numbers in deviant sequences as positive or negative. The models performed well on this subtask, and so had the information needed to complete the quantification task successfully. But they could not use the information in the right way.

Our ICL observations that large or rarely seen numbers can pose representational challenges depend on numerical magnitudes. To see exactly what these correspond to, we used for ICL tasks a linear embedding $Emb : x \mapsto x \cdot W$, for $(x, W) \in \mathbb{R} \times \mathbb{R}^d$. This simple and interpretable mapping preserves the relative ordering of numbers: if $|x| < |y|$ then $||emb(x)|| < ||emb(y)||$, where $|| \cdot ||$ denotes the vector norm. Importantly, this encoding does not introduce boundary effects. Thus, observed boundaries must originate elsewhere.

5.3 The issues with Softmax

We then looked in more detail at the workings of the attention matrix. To recall the basics of attention, let $\mathrm{e}^\ell = (\mathrm{e}^\ell_1, ..., \mathrm{e}^\ell_n)$ be the input embeddings processed by the multi-head attention mechanism at layer ℓ, where e^ℓ_i denotes the embedding of the i-th token in that layer. Each attention head h in layer ℓ is parameterized by Query, Key, and Value weight matrices, denoted as $\mathrm{Q}^{h,\ell}, \mathrm{K}^{h,\ell}$, and $\mathrm{V}^{h,\ell}$, respectively. The dimension d_k corresponds to the embedding size divided by the number of heads. The output of each attention head h in a layer ℓ is a sequence of vectors $(\mathrm{C}^{h,\ell}_1, \mathrm{C}^{h,\ell}_2, ..., \mathrm{C}^{h,\ell}_n)$ where each:

$$\mathrm{C}^{h,\ell}_i = \sum_{j=1}^{n} \left(\mathrm{softmax} \left(\frac{(\mathrm{Q}^{h,\ell}\mathrm{e}^\ell_i)^\top (\mathrm{K}^{h,\ell}\mathrm{e}^\ell_j)}{\sqrt{\mathrm{d}_k}} \right) \right) \mathrm{V}^{h,\ell}\mathrm{e}^\ell_j \tag{3}$$

The primary role of an attention head is to refine the embedding of each input e^ℓ_i by incorporating contextual information from surrounding tokens. However, once the gap between the input values e^ℓ_j in the argument of the Softmax operator in equation 3 surpasses a certain value—specifically, when the gap between the largest value and the others exceeds a threshold—the resulting Softmax weights rapidly saturate. A difference of 4 is typically sufficient. In such cases, the attention weight assigned to the maximum value approaches 1, while the weights for all other values approach 0. As a result, the model focuses almost entirely on a single token, the one associated with the maximum value, while effectively ignoring the rest of the context.[5]

$$\mathrm{C}^{h,\ell}_i = \mathrm{V}^{h,\ell}\mathrm{e}^\ell_{j_0} \tag{4}$$

Importantly, the large values x_j yielding equation 4 come from inputs the model has seldom seen in training. Thus, the Q and K matrices cannot have been trained to handle them.

We verified these predictions by examining the outputs from the attention matrices of the last layer of the model in the quantification task. Figure 2 confirms experimentally the behavior predicted by our analysis: in the case of a significant gap between values, the attention layer puts all the weight on the largest value in the sequence, the value 70; the other elements in the sequence which determined the truth value for (1-a) are ignored, which leads the model to falsely predict the sequence as all positive based only on the large value.

With significant differences in input values, Softmax increasingly resembles a Hardmax function, assigning a weight close to 1 to the largest element and weights near 0 to all others. It also makes the score of negative values tend towards 0 due to the exponential. Significant differences that can affect Softmax occur not only with numerical inputs but also with words processed by classical tokenizers, as seen in the *OpenWebText* corpus (see Figure 8 in Appendix H).

Training on distributions with a much larger range of elements like $D_\mathcal{I} = \mathcal{N}(0, 10)$ or $D_\mathcal{I} = \mathcal{N}(0, 100)$ improved model performance on deviant sequences but significantly increased squared

[5] Our linear embedding function ensures that large differences in number inputs will have the effect noted in equation 4. If $|x|$ is significantly larger than $|y|$, then $||\mathrm{e}(x)||$ will be significantly larger than $||\mathrm{e}(y)||$.

errors on $\mathcal{N}(0, \sigma)$ for small σ. This training gave the Q and K matrices small weights to compensate. Additionally, all models suffered in performance once the out-of-distribution elements x_i in deviant inputs became sufficiently large.

Softmax saturation also adversely affected model performance in the linear functions task. For example, a 12L8AH model's predictions for $f(x) = x$ with this input sequence with $x_1 = 100$: $[100, -1.09, 0.78, 0.26, 0.42]$. The model's predictions are: $[-1.21, -0.28, 2.15, 0.96, 0.65]$. Given this large value, model fails to reasonably approximate the function.[6] This observation follows from the the mathematical nature of Softmax. When a value x_i in the sequence input to the attention mechanism is larger than the other elements of the sequence and other elements in its training, Softmax will assign x_i probability 1 and all other elements in the sequence probability 0. This makes sense in some tasks; a large value in the attention mechanism intuitively signals a strong statistical correlation in context sensitive aspects of meaning (Asher, 2011); Softmax amplifies this value. However, in tasks like ours this is problematic.[7] An input with a large norm representing a large number does not necessarily have a disproportionately greater effect. For our tasks, the model must look at many tokens in the context; with deviant sequences, Softmax prevents the models from doing this.

6 Exploring Alternatives to Softmax

We explored whether alternatives could fix Softmax's deficiencies. We tried temperature scaling and used Softmax with $\tau \in \{5, 10, 20, 50, 100\}$. Table 1 shows this did not resolve the issue on the Linear Function task. We also investigated other proposed alternative mechanisms— Sparsemax (Martins and Astudillo, 2016) and Entmax (Peters et al., 2019), as well as CosFormer (Qin et al., 2022) and SA-Softmax (Zheng et al., 2025). Finally, we evaluated hybrid schemes combining different scoring functions (details in Appendix C). None of these approaches yielded improvements over standard Softmax on our tasks.

7 Solution: Signed scaled averaging (SSA)

For tasks like our ICL tasks requiring integration of information across many tokens, Softmax saturation is clearly harmful. However, tasks requiring focused attention on specific tokens can benefit from Softmax. We propose, rather than having a one size fits all scoring function, to develop a parameterized family of functions that can interpolate between exponential selectivity and more distributed behaviors, allowing the model during training to select the most appropriate behavior.

We replace the exponential in the attention scoring function with a parametrized form

$$x \mapsto (1 + b|x|)^{\mathrm{sgn}(x)\, n},$$

where $b > 0$ and $n \geq 1$ are trainable parameters.[8] This formulation defines a controlled polynomial scoring function whose growth can approximate, but is not limited to, exponential behavior. For sufficiently large parameter values, it recovers exponential growth as a limiting case. For example, setting $b = \frac{1}{m}$ and $n = m$ yields, for $x \geq 0$, $(1 + bx)^{\mathrm{sgn}(x)n} = \left(1 + \frac{x}{m}\right)^m \xrightarrow[m\to\infty]{} e^x$. At the other extreme, when $b = n = 1$, SSA reduces to a linear map $x \mapsto 1 + x$ for positive inputs. The absolute value ensures non-negative scores, while the signed exponent causes negative inputs to decay smoothly toward zero. Unlike exponential scoring, this decay is polynomial, leading to slower score concentration and reduced sensitivity to large-magnitude inputs.

For a vector $z = (z_1, \ldots, z_p) \in \mathbb{R}^p$, SSA replaces Softmax by

$$\mathrm{SSA}(z)_i = \frac{(1 + b\,|z_i|)^{\mathrm{sgn}(z_i)\, n}}{\sum_{k=1}^{p} (1 + b\,|z_k|)^{\mathrm{sgn}(z_k)\, n}}. \quad (5)$$

Appendix J has a full theoretical analysis of SSA.

7.1 Effects of SSA on our ICL tasks

Substituting SSA for Softmax substantially improves performance in our ICL tasks. All results are with our 12L8AH full transformer model. For the quantification task, SSA yields considerable generalization improvement both in handling longer test sequences and managing deviant inputs with extreme values. The heat map in Figure 4 compares error rates for the *every* task: the SSA model (right) maintains lower error across broader ranges of sequence length and input distribution compared to the Softmax model (left). SSA also improves

[6] Though eventually the model begins to recover and approximate better.

[7] The problem occurs also of course with hardmax.

[8] Plots of representative SSA base functions are shown in Figure 10.

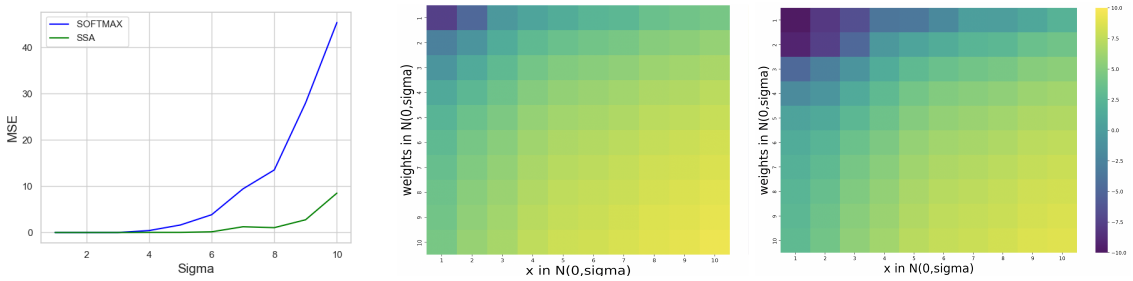

Figure 3: (Left) Comparison plot showing the evolution of MSE for SSA and Softmax-based models (12L8AH) with $D_{\mathcal{F}}, D_{\mathcal{I}}, D_{\mathcal{I}}^{test} \sim \mathcal{N}(0,1)$ and varying $D_{\mathcal{F}}^{test} \sim \mathcal{N}(0,\sigma)$. The heatmap shows the evolution of logarithm of MSE for the Softmax (Middle) and SSA (Right) model when varying both $D_{\mathcal{I}}^{test}$ and $D_{\mathcal{F}}^{test}$.

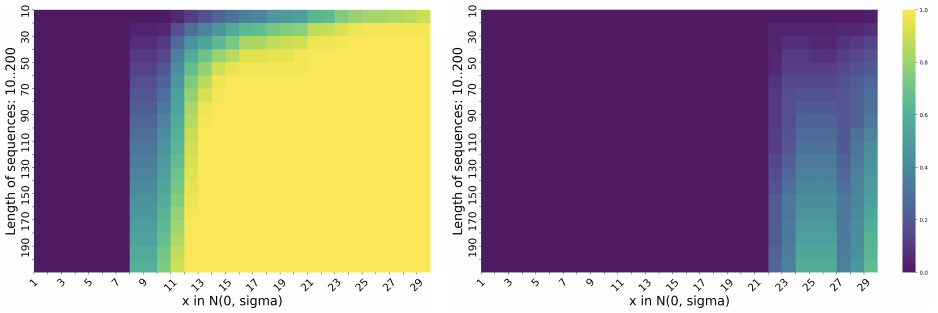

Figure 4: Heatmaps showing the evolution of errors for the 12L8AH model with Softmax (Left) and SSA (Right) on the *every* task. Model was trained on data in $D_{\mathcal{I}} = \mathcal{N}(0,1)$ for lengths from 11 to 40 and tested in $D_{\mathcal{I}}^{test} = \mathcal{N}(0,\sigma)$ for $\sigma \in \{1, ..., 30\}$ and lengths from 10 to 200 for each task. Yellow represents a much higher error rate than purple.

models \ σ	1	2	3	4	5	6	7	8	9	10
SSA	4×10^{-5}	3×10^{-4}	10^{-3}	0.02	0.02	0.15	1.24	1.04	2.74	8.50
Softmax	8×10^{-5}	3×10^{-4}	6×10^{-3}	0.42	1.62	3.84	9.42	13.51	27.99	45.35
Sparsemax	2×10^{-4}	3×10^{-3}	0.28	1.35	3.07	8.73	11.06	29.15	45.39	78.16
Entmax	1×10^{-4}	2.9×10^{-3}	0.27	1.16	2.78	9.62	11.60	25.66	51.30	81.74
$\tau = 5$	3×10^{-4}	1×10^{-2}	0.81	3.17	7.37	16.33	21.43	51.28	74.55	103.71
$\tau = 10$	1×10^{-4}	3×10^{-3}	0.29	1.25	2.98	7.61	10.43	23.43	41.96	68.81
$\tau = 20$	5.2×10^{-5}	2.9×10^{-3}	0.30	1.22	2.97	8.05	11.25	28.48	47.13	78.53
$\tau = 50$	1×10^{-4}	3.1×10^{-3}	0.29	1.29	3.18	8.14	11.68	24.15	42.81	73.51
$\tau = 100$	1×10^{-4}	2.5×10^{-3}	0.26	1.11	2.82	8.53	12.07	29.02	51.43	80.49
SOFT/AVG	7×10^{-5}	3×10^{-3}	0.30	1.22	2.91	7.52	10.32	22.97	38.97	60.03
4 Scoring fts	5×10^{-5}	3×10^{-3}	0.34	1.33	3.18	8.28	10.99	26.31	42.42	70.33
CosFormer	2×10^{-4}	5.6×10^{-3}	0.43	2.07	5.10	12.54	16.91	46.68	66.22	91.02
SA-Softmax	6×10^{-5}	2.8×10^{-3}	0.37	1.83	3.83	11.94	13.72	39.29	58.43	80.31
Llama 3.3 70b	2×10^{-3}	5.50	3.16	13.71	18.21	23.91	28.99	33.51	40.25	48.02

Table 1: Comparison showing the evolution of squared errors for models tested on $x \in D_{\mathcal{I}}^{t} = \mathcal{N}(0,1)$ and weights $a, b \in D_{\mathcal{F}}^{t} = \mathcal{N}(0,\sigma)$. τ represents temperature-scaled Softmax for $\tau \in \{5, 10, 20, 100\}$. All models use 12 layer, 8 attention head full transformer.

performance on the "some" task (see Appendix D). For the linear function learning task, SSA substantially improved performance when tested on out-of-distribution function parameters (Figure 3), while outperforming all alternatives we tested (Table 1).

7.2 SSA with NLP benchmarks

SSA on Decoder-Only models We evaluated the effectiveness of SSA for NLP tasks in several ways. First, we trained GPT-2 small models (124M parameters) from scratch on the FineWebText corpus (10B tokens) for 50,000 steps to compare a standard softmax model against an SSA variant. To assess Model quality, we used perplexity on a separate corpus, *OpenWebText* (Gokaslan and Cohen, 2019). After 50,000 training steps, the SSA model achieved a perplexity of **27.7**, compared to **31.7** for the softmax model, a **12.6% improvement**.

These improvements in perplexity translated into better performance on downstream zero-shot and few-shot benchmarks, including ARC-Challenge, ARC-Easy (Clark et al., 2018), HellaSwag (Zellers et al., 2019), LAMBADA (Paperno et al., 2016), and SuperGLUE (Wang et al., 2019). We used the LM Eval harness (Biderman et al., 2024) for early learning evaluation, following standard practice. The SSA variant consistently outperformed the softmax counterpart across all evaluated tasks (Table 2). Performance gains were most pronounced in linguistically and semantically demanding benchmarks such as RTE (Dagan et al., 2006), WiC (Pilehvar and Camacho-Collados, 2019), and WSC (Levesque, 2012), which require contextual reasoning and relational inference.

Task	Zero-shot		Few-shot (3-shot)	
	Softmax	SSA	Softmax	SSA
LM-Eval Benchmarks				
ARC-Challenge	20.8	**23.2**(+11.5% ↑)	23.40	**25.92**(+10.8% ↑)
ARC-Easy	45.5	**47.9**(+5.3% ↑)	48.13	**50.19**(+4.3% ↑)
HellaSwag	29.2	**29.5**(+1% ↑)	30.33	**31.25**(+3.2% ↑)
LAMBADA	24.8	**25.4**(+6.5% ↑)	15.44	**16.46**(+6.5% ↑)
SuperGLUE Benchmarks				
BoolQ	60.2	**61.1**(+1.5% ↑)	55.34	**57.08**(+2.7% ↑)
CB	22.5	**58.5**(+160% ↑)	37.94	**51.34**(+35.3% ↑)
COPA	62.3	**64.9**(+4.2% ↑)	63.31	**72.69**(+14.8% ↑)
MultiRC	56.4	**57.9**(+2.7% ↑)	53.36	**54.80**(+2.7% ↑)
RTE	45.7	**56.8**(+24.3% ↑)	47.17	**53.19**(+12.7% ↑)
WiC	48.0	**52.0**(+8.3% ↑)	43.80	**47.74**(+8.3% ↑)
WSC	31.8	**41.3**(+29.9% ↑)	35.55	**45.21**(+27.2% ↑)

Table 2: Zero- and few-shot performance of GPT-2 models (124M) trained from scratch on FineWebText for 50k steps, comparing Softmax and SSA across LM-Eval and SuperGLUE benchmarks. robustness is reported in Table 4

SSA on Encoder-Only models We trained variants of BabyBERTa (Huebner et al., 2021) on the AO-CHILDES corpus, comparing the standard Softmax baseline with SSA using fixed exponents $n = 1.5$ and $n = 2$. BabyBERTa is a compact RoBERTa-style encoder trained on child-directed speech. It is well-suited for probing small models' ability to capture grammatical dependencies in limited-data settings. We evaluated the models on a set of linguistic probes from (Huebner et al., 2021) using the masked language modeling (MLM) metric, which measures token-level accuracy against distractors. As shown in Table 3, SSA improves BabyBERTa's performance across a range of grammatical phenomena. SSA-2 achieves the strongest gains on syntax-sensitive tasks such as subject–verb agreement across relative clauses and argument structure alternations, while SSA-1.5 yields improvements on morphological and lexical tests such as irregular verbs and pronoun gender.

Overall, these results indicate that SSA enhances both intrinsic modeling efficiency and downstream task performance of decoder-only models, while also enhancing the grammatical sensitivity of encoder-only models, as well as g our observations for decoder-only models. This supports SSA's potential as a drop-in replacement for Softmax in large-scale language modeling.

linguistic probe	Softmax	SSA 1.5	SSA 2
agr_subj_verb-across_PP	56	58.95	**65.95**
agr_subj_verb-across_RC	55.5	57.7	**61.55**
agr_subj_verb-in_Q+aux	76.5	**79.0**	70.45
anaphor_agr-pron_gender	48.1	51.45	**53.7**
arg_str-dropped_arg	79.65	74.65	**85.55**
arg_str-swapped_args	83.3	**92.0**	88.0
arg_str-transitive	53.44	53.85	**57.2**
binding-principle_a	78.25	**87.9**	80.2
case-subjective_pron	85.55	89.7	**91.75**
filler-gap-wh_Q_subject	79.2	**83.3**	68.25
irregular-verb	70.05	**78.3**	69.2
quantifiers-superlative	71.2	**83.95**	65.25

Table 3: Performance of BabyBERTa models trained from scratch on AO-CHILDES using Softmax and SSA (1.5, 2), evaluated on linguistic probes from (Huebner et al., 2021) with the MLM metric. Agr: agreement; arg_str: argument structure; Subj: subject; Pron: pronoun; PP: prepositional phrase; RC: relative clause; Det: determiner; N: noun; arg: argument.

8 Conclusion

Transformer models struggle to generalize effectively to out-of-distribution data. We identified the Softmax scoring function in the attention mechanism as a factor contributing to this challenge and introduced SSA, a novel scoring method that significantly improves the performance on both mathematical and NLP tasks.

SSA enhances a model's ability to capture linguistic structure and allocate attention more effectively, addressing softmax's tendency to saturate and focus narrowly on a few tokens. By parametrically controlling attention, SSA improves generalization without increasing model complexity or reducing accuracy. While it does not solve all generalization challenges, SSA directly mitigates those caused by softmax's inflexible scoring, offering a simple yet effective drop-in alternative for transformers' attention mechanism.

Limitations

Due to hardware and data constraints, we were unable to scale models beyond 124M parameters. We trained a GPT-2 model (124M parameters) from scratch on the FineWebText dataset for 50,000 steps, which required approximately 120 GPU-hours on an A100 80GB GPU. To enable a rigorous comparison, we trained both a Softmax and an SSA version of the model, resulting in a total of over 240 GPU-hours of training.

While SSA provides noticeable improvements in generalization, it does not fully address all the shortcomings of the attention mechanism. In particular, our heatmaps indicate that SSA struggles to generalize in scenarios where both the input x_i and the test-time function distribution $D_{\mathcal{F}}^{\text{test}}$ diverge significantly from the training distribution $D_{\mathcal{F}}$.

Thus, SSA does not address the fact that, as we have noted above, the simple mathematical structure of the attention mechanism conflates the value of tokens with their importance for the particular task.

References

Nicholas Asher. 2011. *Lexical Meaning in Context: A web of words*. Cambridge University Press.

Nicholas Asher, Swarnadeep Bhar, Akshay Chaturvedi, Julie Hunter, and Soumya Paul. 2023. Limits for learning with large language models. In *12th Joint Conference on Lexical and Computational Semantics (*Sem)*. Association for Computational Linguistics.

Satwik Bhattamishra, Arkil Patel, Phil Blunsom, and Varun Kanade. 2023. Understanding in-context learning in transformers and llms by learning to learn discrete functions. *arXiv preprint arXiv:2310.03016*.

Stella Biderman, Hailey Schoelkopf, Lintang Sutawika, Leo Gao, Jonathan Tow, Baber Abbasi, Alham Fikri Aji, Pawan Sasanka Ammanamanchi, Sidney Black, Jordan Clive, Anthony DiPofi, Julen Etxaniz, Benjamin Fattori, Jessica Zosa Forde, Charles Foster, Jeffrey Hsu, Mimansa Jaiswal, Wilson Y. Lee, Haonan Li, and 11 others. 2024. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*. Describes LM-evaluation harness and best practices for LM evaluation.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Akshay Chaturvedi, Swarnadeep Bhar, Soumadeep Saha, Utpal Garain, and Nicholas Asher. 2024. Analyzing Semantic Faithfulness of Language Models via Input Intervention on Question Answering. *Computational Linguistics*, pages 1–37.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché-Buc, editors, *Machine Learning Challenges — Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Textual Entailment (MLCW 2005), Lecture Notes in Computer Science, Vol.3944*, page 177–190. Springer-Verlag Berlin Heidelberg.

P Kingma Diederik. 2014. Adam: A method for stochastic optimization. *(No Title)*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, and 1 others. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Shivam Garg, Dimitris Tsipras, Percy S Liang, and Gregory Valiant. 2022. What can transformers learn in-context? a case study of simple function classes. *Advances in Neural Information Processing Systems*, 35:30583–30598.

Angeliki Giannou, Liu Yang, Tianhao Wang, Dimitris Papailiopoulos, and Jason D Lee. 2024. How well can transformers emulate in-context newton's method? *arXiv preprint arXiv:2403.03183*.

Aaron Gokaslan and Vanya Cohen. 2019. Openwebtext corpus. `https://skylion007.github.io/OpenWebTextCorpus/`.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R Bowman. 2019. Do attention heads in bert track syntactic dependencies? *arXiv preprint arXiv:1911.12246*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Philip A Huebner, Elior Sulem, Fisher Cynthia, and Dan Roth. 2021. Babyberta: Learning more grammar with small-scale child-directed language. In *Proceedings of the 25th conference on computational natural language learning*, pages 624–646.

Hector J. Levesque. 2012. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*.

Siddarth Mamidanna, Daking Rai, Ziyu Yao, and Yilun Zhou. 2025. All for one: Llms solve mental math at the last token with information transferred from other tokens. *arXiv preprint arXiv:2509.09650*.

Andre Martins and Ramon Astudillo. 2016. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pages 1614–1623. PMLR.

R Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D Hardy, and Thomas L Griffiths. 2024. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, 121(41):e2322420121.

Omar Naim and Nicholas Asher. 2024. Reexamining learning linear functions in context. ArXiv:2411.11465 [cs.LG].

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, and 1 others. 2022. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, and 1 others. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849.

Ben Peters, Vlad Niculae, and André FT Martins. 2019. Sparse sequence-to-sequence models. *arXiv preprint arXiv:1905.05702*.

Mohammad Taher Pilehvar and Jose Camacho-Collados. 2019. Wic: the word-in-context dataset for evaluating context-sensitive meaning representations. In *NAACL-HLT 2019 (Workshop)*.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. 2022. cosformer: Rethinking softmax in attention. In *International Conference on Learning Representations*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, and 1 others. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Allan Raventós, Mansheej Paul, Feng Chen, and Surya Ganguli. 2024. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. *Advances in Neural Information Processing Systems*, 36.

Pradyut Sekhsaria, Marcel Mateos Salles, Hai Huang, and Randall Balestriero. 2025. Lora users beware: A few spurious tokens can manipulate your finetuned model. *arXiv preprint arXiv:2506.11402*.

Yuandong Tian, Yiping Wang, Beidi Chen, and Simon S Du. 2023. Scan and snap: Understanding training dynamics and token composition in 1-layer transformer. *Advances in neural information processing systems*, 36:71911–71947.

Jesse Vig. 2019. Visualizing attention in transformer-based language representation models. *arXiv preprint arXiv:1904.02679*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint arXiv:1905.00537*.

Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. 2021. An explanation of in-context learning as implicit bayesian inference. *arXiv preprint arXiv:2111.02080*.

Jiacheng Ye, Jiahui Gao, Shansan Gong, Lin Zheng, Xin Jiang, Zhenguo Li, and Lingpeng Kong. 2024. Beyond autoregression: Discrete diffusion for complex reasoning and planning. *arXiv preprint arXiv:2410.14157*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy.

Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. 2024. Trained transformers learn linear models in-context. *Journal of Machine Learning Research*, 25(49):1–55.

Chuanyang Zheng, Yihang Gao, Guoxuan Chen, Han Shi, Jing Xiong, Xiaozhe Ren, Chao Huang, Xin Jiang, Zhenguo Li, and Yu Li. 2025. Self-adjust softmax. *arXiv preprint arXiv:2502.18277*.

A Training details

A.1 ICL Tasks

Our ICL tasks involve training from scratch on sequences containing in-context examples (input-output pairs) $(x_1, f(x_1), ..., x_i)$ ending with a query input x_i that is used to generate the corresponding output.

Model Architecture. We use a decoder-only Transformer architecture from the GPT-2 family (Radford et al., 2019) with 12 layers, 8 attention heads, and a 256-dimensional embedding space. Dropout is set to 0 as we sample fresh prompts at each training step. The model takes as input a sequence of vectors in its embedding space and predicts the next vector in the sequence.

Input/Output Encoding. Both prompt inputs and outputs are then mapped into the model's latent embedding space of dimension 256 through a learnable linear transformation $W_{\text{enc}} \in \mathbb{R}^{256}$. The model processes this sequence and outputs vectors in the same embedding space. These output vectors are mapped back to scalar predictions via a separate learnable linear transformation $W_{\text{dec}} \in \mathbb{R}^{256}$ (implemented as a dot product).

Training Procedure. Models are trained for 500k steps using the Adam optimizer (Diederik, 2014) with a learning rate of 10^{-4} and batch size of 64. At each training step, we sample a fresh batch of random prompts by: (1) sampling a random function g from the function class according to $\mathcal{D}_{\mathcal{F}}$, (2) sampling inputs $x_1, \ldots, x_{k+1}$ independently from $\mathcal{D}_{\mathcal{I}}$, and (3) evaluating g on these inputs to produce the prompt. For each prompt, the loss is computed as $\frac{1}{k}\sum_{i=1}^{k}(\hat{y}_i - g(x_i))^2$ where $\hat{y}_i$ is the model's prediction.

Curriculum Learning. We conduct training both with and without curriculum learning. When employing curriculum learning, we train on a set S of training sequences of varying lengths, ranging from 1 to $k = 40$. Specifically, we start with prompt length 3 (number of input-output pairs). Every 2,000 training steps, we increase the length by 2, until reaching the full prompt length.

Additional training information: We use the Adam optimizer (Diederik, 2014) , and a learning rate of 10^{-4} for all models.

Computational resources: We used Nvidia A-100 GPUs to train the different versions of transformer models from scratch, with an average training time of 4 hours and used Nvidia Volta (V100 - 7,8 Tflops DP) GPUs for the fine-tuning of LLaMA 3.1 8B involved in these experiments.

Fine-tuning LLaMA 3.1 8B was fine tuned on 26000 randomly generated sequences S progressing from 1 to 40 for 2 epochs using LoRA (Hu et al., 2022). The input values were drawn from $\mathcal{N}(0, 1)$. At test time, we were only able to run a few sequences for each possible we examined sequence lengths from 10 to 200 and number distributions from $\mathcal{N}(0, \sigma)$ for $1 \leq \sigma \leq 30$. This meant that we could only run a few test batches, since we need to look at 600 total pairs for each task. We averaged the prediction errors on the each $(S^{\text{test}}, D_{\mathcal{I}}^{\text{test}})$ possibility.

For our attempted fine-tuning of LLaMA 3.1 8B on the function task, we set the input sequence to be of the form $(x_1, f(x_1), x_2, f(x_2), ..., x_n)$ requiring that the output be of the form $f(x_n)$, a single numerical value. The model returned a list of values $(w_1, w2, ...)$. It failed to capture the basic input and output pattern.

A.2 Training on FineWebText

Training Details. We trained a GPT-2–style Transformer decoder model from scratch on approximately 10B tokens from the FineWebText corpus. The model consists of 12 Transformer layers with 12 self-attention heads per layer, a hidden dimension of 768, and a maximum context length of 1024 tokens. Training was performed using the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a weight decay of 0.1. We used a peak learning rate of 6×10^{-4} with linear warmup followed by cosine decay to a minimum learning rate of 6×10^{-5} over 50k training iterations. The micro-batch size was 12 sequences per GPU, with gradient accumulation used to obtain a larger effective batch size across workers. Models were trained with a standard causal language modeling objective using next-token prediction. Training was conducted using PyTorch Distributed Data Parallel (DDP) across 12 NVIDIA A100 GPUs with 80GB memory each, with mixed-precision training and TF32 enabled for matrix multiplications. Checkpoints were saved periodically to allow training resumption and evaluation.

Aside from the choice of scoring function and its parameterization, all architectural components, optimization settings, and training procedures were kept identical across models. As a result, we observe no meaningful difference in training time between SSA- and Softmax-based models.

B SSA settings

For each task, we trained two classes of models differing only in the scoring function. The first uses

the standard Softmax-based attention mechanism, while the second replaces Softmax with *SSA*. In the SSA setting, a small number of additional parameters are learned independently for each attention head, such that every head in every Transformer layer has its own set of SSA parameters. Concretely, SSA introduces only two scalar parameters per attention head. For a model with 12 layers and 12 attention heads per layer, this corresponds to a total of $2 \times 12 \times 12 = 288$ additional parameters, which is negligible compared to the overall model size. We initialize n to 1.5 and parameterize b to 1. All remaining initialization and optimization details required for exact reproducibility are provided in the released code. Aside from the choice of scoring function and its parameterization, all architectural components, optimization settings, and training procedures were kept identical across models. As a result, we observe no meaningful difference in training time between SSA- and Softmax-based models.

C Using mixtures of scoring functions

A natural test is to take temperature-scaled Softmax with parameter τ to rescale the exponential behavior. We tried several scaling factors $\tau \in \{5, 10, 20, 50, 100\}$, but none produced better results than the standard Softmax. In addition, we experimented with alternative normalization functions that yield sparse attention distributions, including Sparsemax (Martins and Astudillo, 2016) and Entmax (Peters et al., 2019). These methods replace the Softmax normalization while preserving the overall attention framework. However, neither Sparsemax nor α-Entmax led to performance improvements on our tasks.

We next partitioned the attention heads such that half utilized Softmax-based scoring, while the remaining half employed uniform averaging over all tokens. This design, we thought, would preserve contextual breadth, reduce the risk of focusing to specific tokens, and also increase the model's expressiveness through multiple scoring functions. We experimented with four distinct known scoring functions (tanh, average, ReLU, and x^2), assigning two heads to each (For detailed results see Table 1). This approach improved over than Softmax on $\mathcal{N}(0, 1)$ but was less good elsewhere.

We additionally tested COSFORMER (Qin et al., 2022), which replaces the exponential weighting in Softmax with a cosine-modulated kernel combined with linear normalization. But, cosFormer did not improve performance on our ICL tasks and often underperformed the Softmax baseline (Table 1).

Finally, we experimented with the recently proposed Self-Adjusting Softmax (SA-Softmax) (Zheng et al., 2025). However, this mechanism did not yield better generalization or accuracy than the standard Softmax attention (Table 1).

D Heat map for the "some" task with SSA for model trained from scratch

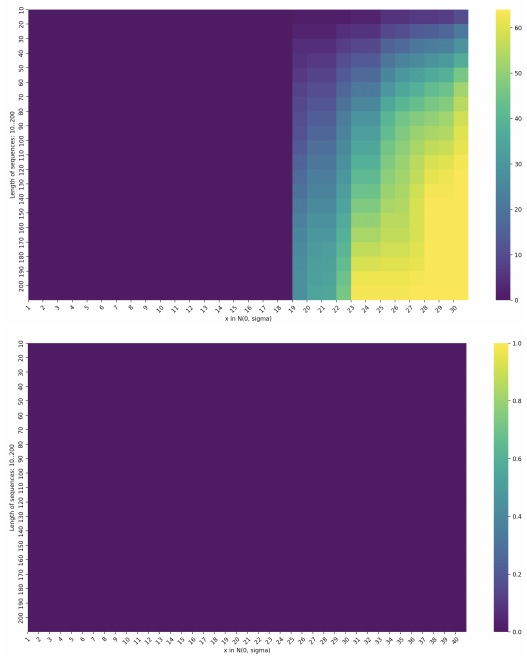

Figure 5: Heatmap showing the evolution of errors for the task *some* trained on data in $D_{\mathcal{I}} = \mathcal{N}(0, 1)$ for lengths from 11 to 40 and tested in $D_{\mathcal{I}}^{\text{test}} = \mathcal{N}(0, \sigma)$ for $\sigma \in \{1, ..., 30\}$ and lengths from 10 to 200. The first figure is for the Softmax-based model and the second with SSA.

E Heat map for the "some" task with pre-trained and finetuned models

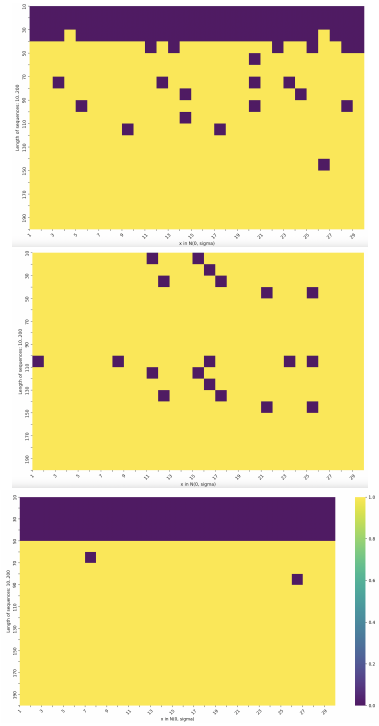

Figure 6: Heatmap showing the evolution of errors for the task "some" on different models "Pre-trained Llama 3.3 70B", "Pre-trained Llama 3.1 8B" and finetuned on Llama 31 8B", respectively from top to bottom on data in $D_{\mathcal{I}} = \mathcal{N}(0, 1)$ for lengths from 11 to 40 and tested in $D^t_{\mathcal{I}} = \mathcal{N}(0, \sigma)$ for $\sigma \in \{1, ..., 10\}$ and lengths from 10 to 200.

F Heat map for the "every" task with pre-trained and finetuned models

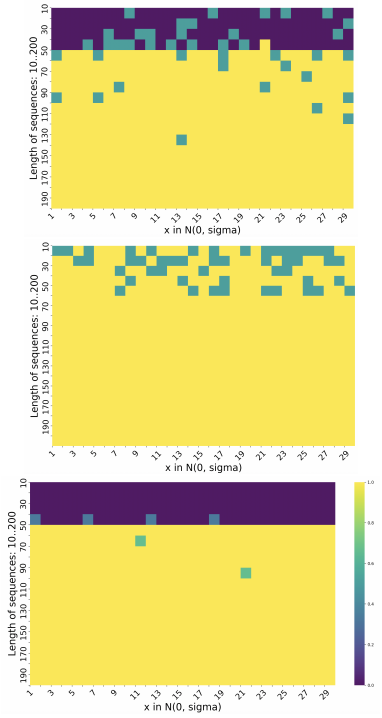

Figure 7: Heatmap showing the evolution of errors for the task "every" on "Pre-trained Llama 3.3 70B" (First), "Pre-trained Llama 3.1 8B" (Second) with 5 shot learning and finetuned on Llama 31 8B" (Third), on data in $D_{\mathcal{I}} = \mathcal{N}(0, 1)$ for lengths from 1 to 40 and tested in $D^t_{\mathcal{I}} = \mathcal{N}(0, \sigma)$ for $\sigma \in \{1, ..., 10\}$ and lengths from 10 to 200.

G Additional Results and Robustness

Task	Zero-shot		Few-shot (3-shot)	
	Softmax	SSA	Softmax	SSA
LM-Eval Benchmarks				
ARC-Challenge	20.8 ± 1.21	$\mathbf{23.2 \pm 1.21}$	23.40 ± 1.20	$\mathbf{25.92 \pm 1.22}$
ARC-Easy	45.5 ± 1.02	$\mathbf{47.9 \pm 1.03}$	48.13 ± 1.03	$\mathbf{50.19 \pm 1.03}$
HellaSwag	29.2 ± 0.46	$\mathbf{29.5 \pm 0.45}$	30.33 ± 0.46	$\mathbf{31.25 \pm 0.45}$
LAMBADA	24.8 ± 0.61	$\mathbf{25.4 \pm 0.60}$	15.44 ± 0.54	$\mathbf{16.46 \pm 0.51}$
SuperGLUE Benchmarks				
BoolQ	60.2 ± 0.85	$\mathbf{61.1 \pm 0.86}$	55.34 ± 0.85	$\mathbf{57.08 \pm 0.87}$
CB	22.5 ± 6.09	$\mathbf{58.5 \pm 6.74}$	37.94 ± 6.74	$\mathbf{51.34 \pm 6.70}$
COPA	62.3 ± 4.73	$\mathbf{64.9 \pm 4.92}$	63.31 ± 4.82	$\mathbf{72.69 \pm 4.69}$
MultiRC	56.4 ± 0.71	$\mathbf{57.9 \pm 0.71}$	53.36 ± 0.71	$\mathbf{54.80 \pm 0.72}$
RTE	45.7 ± 3.01	$\mathbf{56.8 \pm 3.00}$	47.17 ± 3.00	$\mathbf{53.19 \pm 3.01}$
WiC	48.0 ± 1.98	$\mathbf{52.0 \pm 1.98}$	43.80 ± 1.98	$\mathbf{47.74 \pm 1.97}$
WSC	31.8 ± 4.74	$\mathbf{41.3 \pm 4.74}$	35.55 ± 4.89	$\mathbf{45.21 \pm 4.83}$

Table 4: Zero- and few-shot performance of GPT-2 models (124M) trained from scratch on FineWebText for 50k steps, comparing Softmax and SSA across LM-Eval and SuperGLUE benchmarks. Reported values are mean $\pm$ standard error across seeds.

H Repartition of token norms in the OpenWebText

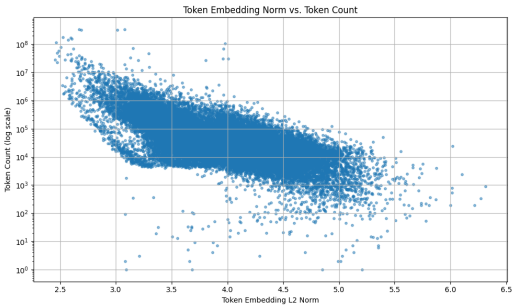

Figure 8: Plot showing how many tokens have a norm of value x, for OpenWebText using GPT-2 Tokenizer. The x-axis is token embedding norm and the y-axis is token count (log scale)

I Example showing that Boundary values behaviors

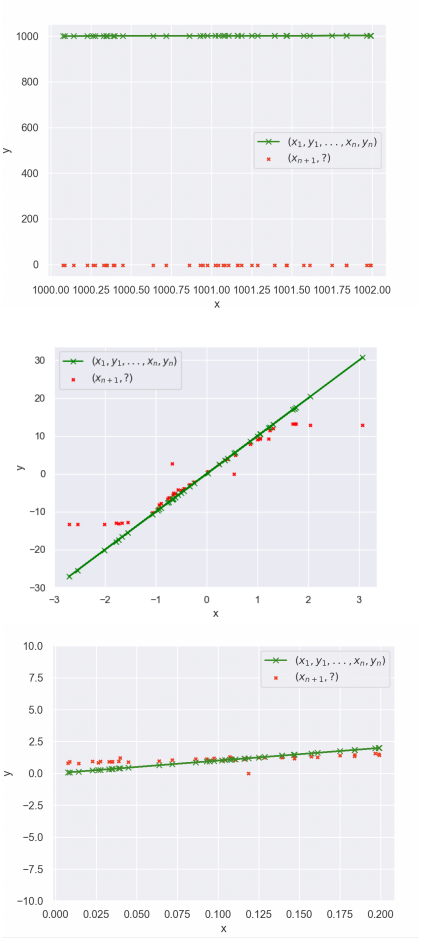

Figure 9: Plots for model 12L8AH, trained on $D_{\mathcal{I}}, D_{\mathcal{F}} \sim \mathcal{N}(0, 1)$ for $f(x) = x$ for high values (First) of x and $f(x) = 10x$ for normal (Second) then for low values of x (Third)

models \ σ	1	10	20	30	40	50	60	70	80	90	100
12L8AH	6.4e-05	0.05	0.25	0.50	0.82	1.21	1.66	1.87	2.02	2.14	2.20

Table 5: Table showing the influence of deviant inputs: Comparison showing the evolution of squared errors for models tested on $x \in D_{\mathcal{I}}^t = \mathcal{N}(0,1)$, except the first element $x_0 \in D_{\mathcal{I}}^t = \mathcal{N}(0,\sigma)$ and weights $a, b \in D_{\mathcal{F}}^t = \mathcal{N}(0,1)$.

models \ σ	1	2	3	4	5	6	7	8	9	10
1L1AH	0.1	0.8	5.1	13.1	26.9	39.7	53.0	84.8	120.0	153.2
1L2AH	0.1	0.8	5.3	14.4	29.8	41.1	55.0	93.8	120.4	159.2
1L4AH	0.0	0.2	2.7	8.7	19.9	32.0	42.8	64.5	92.3	131.2
2L1AH	0.0	0.1	2.0	4.9	13.7	27.0	36.1	64.9	99.0	134.0
2L2AH	0.0	0.0	1.6	3.2	9.3	25.5	32.0	61.1	92.9	127.8
2L4AH	0.0	0.0	0.9	2.6	7.5	19.3	27.3	51.8	90.2	119.4
3L1AH	0.0	0.0	0.9	3.0	8.2	16.8	24.4	48.4	76.7	113.2
3L2AH	0.0	0.0	0.7	2.3	6.5	15.9	22.5	43.1	74.0	102.5
3L4AH	0.0	0.0	0.6	1.9	5.5	13.8	20.4	42.2	70.3	100.4
6L4AH	0.0	0.0	0.5	1.6	4.6	11.6	16.8	33.7	58.3	87.9
12L8AH	0.0	0.0	0.3	1.1	2.9	7.9	11.9	28.3	46.9	73.5
18L8AH	0.0	0.0	0.2	1.1	2.8	7.1	10.3	22.9	40.3	64.6
$2Al32AH$	1.17	2.64	3.47	5.01	7.88	16.85	24.1	40.98	66.04	95.03
$12Al8AH$	0.0	0.0	0.41	1.70	3.92	10.40	14.04	30.20	52.69	79.13

Table 6: Comparison to show the evolution of squared error over different models. $D_{\mathcal{I}}^{test} \sim N(0,1)$.

J Theoretical Analysis of SSA

In this section we will present a theoretical analysis of SSA and compares its structural behavior with the exponential function underlying the softmax.

We analyze growth, gradients, effective slope, Lipschitzness, ratios, entropy, Jacobian stability, and identify the conditions that prevent hardmax-like collapse.

Let's consider:

$$f(x) = (1 + b|x|)^{\operatorname{sgn}(x)\, n}, \qquad b > 0,\ n \geq 1, \tag{6}$$

Logits are denoted $s_1, \ldots, s_k \in \mathbb{R}$.

$$\alpha_i^{\text{SSA}} = \frac{f(s_i)}{\sum_{j=1}^{k} f(s_j)}. \tag{7}$$

J.1 Growth Behavior: SSA vs. Exponential

Softmax uses the exponential function, whose super-polynomial growth causes severe sensitivity to logit differences:

$$e^x \to \infty \quad \text{exponentially in } x.$$

SSA behaves fundamentally differently. Since $\operatorname{sgn}(x)$ determines the exponent, SSA admits the piecewise form

$$f(x) = \begin{cases} (1+bx)^n, & x \geq 0, \\ (1+b|x|)^{-n}, & x < 0. \end{cases}$$

Hence the asymptotics are

$$x \to +\infty: \quad f(x) \sim (bx)^n,$$
$$x \to -\infty: \quad f(x) \sim (b|x|)^{-n}.$$

Thus f grows only polynomially, in stark contrast to the exponential. It therefore reacts more gently to large positive logits.

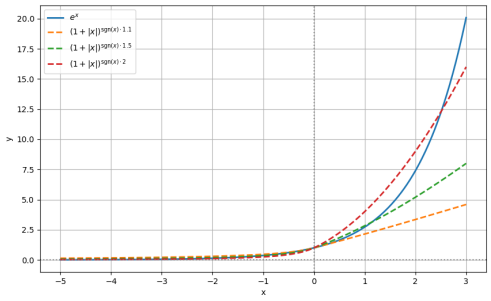

Figure 10: Illustration of the base function $(1 + b|x|)^{\operatorname{sgn}(x)n}$ used in SSA, plotted for $b = 1$ and $n \in \{1.1, 1.5, 2\}$. The curves demonstrate that SSA exhibits behavior similar to the exponential function, but with a tunable growth rate toward $+\infty$, which increases with larger values of the exponent n.

J.2 Gradient Structure and Effective Slope

Piecewise derivative. Differentiation yields

$$f'(x) = \begin{cases} nb(1+bx)^{n-1}, & x > 0, \\ nb(1+b|x|)^{-n-1}, & x < 0, \end{cases} \qquad f'(0) = nb.$$

Therefore,

$$|f'(x)| = \begin{cases} \Theta(|x|^{n-1}), & x \to +\infty, \\ \Theta(|x|^{-n-1}), & x \to -\infty. \end{cases}$$

Softmax satisfies $f'(x) = e^x$, which diverges exponentially for large x.

Effective slope. Define $g(x) = \frac{f'(x)}{f(x)}$. From the expressions above,

$$g(x) = \frac{nb}{1 + b|x|},$$

with $g(0) = nb$ and $\lim_{|x|\to\infty} g(x) = 0$.

Thus SSA exhibits *self-regularizing gradient suppression*: extremely large logits exert vanishing influence, unlike softmax where $g(x) \equiv 1$.

J.3 Ratio Growth, Concentration, and Entropy

Let $s_1 \geq s_2$ denote the two largest logits.

Softmax ratio.

$$\frac{\alpha_1^{\text{exp}}}{\alpha_2^{\text{exp}}} = e^{s_1 - s_2},$$

which grows exponentially in the gap $\Delta = s_1 - s_2$, yielding rapid collapse toward a one-hot distribution.

SSA ratio (for $s_1, s_2 \geq 0$).

$$\frac{\alpha_1^{\text{SSA}}}{\alpha_2^{\text{SSA}}} = \left(\frac{1 + bs_1}{1 + bs_2}\right)^n = \left(1 + \frac{b\Delta}{1 + bs_2}\right)^n.$$

This ratio grows only polynomially in Δ. Consequently, SSA avoids the aggressive concentration typical of softmax.

Entropy lower bound. Given any upper bound $\alpha_{\max} < 1$, the Shannon entropy satisfies

$$H(\alpha) \geq -\alpha_{\max} \log \alpha_{\max} - (1-\alpha_{\max}) \log\left(\frac{1 - \alpha_{\max}}{k-1}\right)$$

In Section J.6, we prove that $\alpha_{\max}$ is uniformly bounded away from 1 for finite n under bounded logits.

J.4 Lipschitz Continuity of SSA Scores

On a bounded interval $|x| \leq M$,

$$|f'(x)| \leq nb(1 + bM)^{n-1},$$

so f is Lipschitz on compact domains with polynomial Lipschitz constant

$$L_{\text{SSA}}(M) = nb(1 + bM)^{n-1}.$$

For softmax,

$$L_{\exp}(M) = \sup_{|x| \leq M} e^x = e^M,$$

which grows exponentially with M.

J.5 Jacobian of SSA Attention

The normalized weights satisfy $\alpha_i = f_i/S$ with $S = \sum_j f_j$. Thus,

$$\frac{\partial \alpha_i}{\partial s_m} = \frac{\mathbf{1}_{i=m} f_i'}{S} - \frac{f_i f_m'}{S^2}.$$

Under the bounded-logit assumption $|s_i| \leq M$,

$$f_i \in [(1+bM)^{-n}, (1+bM)^n], \qquad |f_i'| \leq nb(1+bM)^{n-1}.$$

These bounds imply that all Jacobian entries remain polynomially bounded in M, b, and n, in contrast to softmax whose Jacobian magnitudes can grow exponentially with logit variance.

J.6 Conditions Preventing Hardmax-Like Collapse

A normalized scoring mechanism collapses to a hardmax when

$$\frac{f(s_{\max})}{\sum_{j \neq \max} f(s_j)} \to \infty.$$

Softmax. For softmax,

$$\frac{e^{s_{\max}}}{e^{s_j}} = e^{s_{\max} - s_j},$$

so even constant positive gaps cause divergence, making collapse unavoidable.

SSA under bounded logits. Assume $|s_i| \leq M$. Then

$$f(s_i) \in [(1 + bM)^{-n}, (1 + bM)^n],$$

and thus

$$\alpha_{\max}^{\text{SSA}} \leq \frac{(1 + bM)^n}{(1 + bM)^n + (k-1)(1 + bM)^{-n}}.$$

Define

$$\varepsilon = \frac{(k-1)(1 + bM)^{-n}}{(1 + bM)^n + (k-1)(1 + bM)^{-n}} > 0.$$

Hence

$$\alpha_{\max}^{\text{SSA}} \leq 1 - \varepsilon < 1.$$

Theorem 1 (No Hardmax Collapse for SSA with Finite n). *Let $b > 0$, $n < \infty$, and suppose logits satisfy $|s_i| \leq M$. Then*

$$\max_i \alpha_i^{\mathrm{SSA}} \leq 1 - \varepsilon,$$

with $\varepsilon > 0$ given above. Thus SSA cannot *collapse to a one-hot distribution for finite n unless $|s_i| \to \infty$ or $n \to \infty$.*

Therefore, unlike softmax, SSA inherently resists overconcentration over any bounded logit range.

J.7 Summary

SSA exhibits several structural advantages over softmax:

- **Polynomial** growth and decay versus exponential growth.
- **Self-regularizing gradients** with effective slope $g(x) = nb/(1 + b|x|)$.
- **Entropy preservation**: probabilities remain bounded away from 1.
- **Polynomial Lipschitz constants** for both scores and Jacobians.
- **Provable non-collapse**: hardmax-like behavior is impossible for finite n under bounded logits.

These properties explain the improved numerical stability, robustness, and reduced overconfidence provided by SSA in attention mechanisms.

K Prompts used for tests on pre-trained models

"Linear Function" System Prompt

You are an auto-regressive AI model designed to predict the next value in a sequence of input-output pairs that follow a linear function $f(x) = ax + b$. Your task is to analyze the given input-output pairs and predict the output for the final input value.
Here are some examples to illustrate the task:
Example 1:
CONTEXT: [1, 3, 2, 5, 3, 7, 4]
#Answer: 9
Example 2:
CONTEXT: [0, 1, 2, 5, 4, 9, 6]
#Answer: 13
Example 3:
CONTEXT: [1, -1, 2, -3, 3, -5, 4]
#Answer: -7
Example 4:
CONTEXT: [0, 0, 2, 6, 4, 12, 6]
#Answer: 18
Example 5:
CONTEXT: [-2, -3, 0, 1, 2, 5, 4]
#Answer: 9
Now, given a new sequence of input-output pairs where the last output is missing, predict the final value.
IMPORTANT:

- DO NOT include any explanations in your response.
- DO NOT use any PYTHON code in your response.
- GIVE JUST THE NUMERICAL OUTPUT AS THE ANSWER.

"AND" Task System Prompt

You are an auto-regressive AI model designed to evaluate whether each sublist of a list of numbers is entirely positive. Your task is to process the list incrementally, verifying the positivity of each sublist one by one. A sublist is considered "TRUE" if all its elements are greater than zero. Once a sublist contains a non-positive number, all subsequent sublists will be marked as "FALSE". Although you process the list step-by-step, you will only output the final list of booleans once all sublists have been evaluated.
Here are some examples to illustrate the task:
Example 1:
CONTEXT: [1, 1, 2, 3, -1, 2, 1]
#Answer: [True, True, True, True, False, False, False]
Example 2:
CONTEXT: [0.1, -9, -0.11, 5, 0, 3.5]
#Answer: [True, False, False, False, False, False]
Example 3:
CONTEXT: [-1, -2, -3, -4, -5]
#Answer: [False, False, False, False, False]
Example 4:
CONTEXT: [0.5, 1.5, -0.5, 2.5, -2.5]
#Answer: [True, True, False, False, False]
Example 5:
CONTEXT: [10, -10, 0, 0.01, -0.01]
#Answer: [True, False, False, False, False]
Now, given a new list of numbers, perform the same task and provide the final output in the specified format.
IMPORTANT:

- DO NOT include any other text in your response.
- DO NOT use any PYTHON code in your response.
- GIVE JUST THE OUTPUT LIST AS THE ANSWER.

"OR" Task System Prompt

You are an auto-regressive AI model designed to evaluate whether each sublist of a list of numbers has a positive element. Your task is to process the list incrementally, verifying if there exists a positive element in each sublist one by one. A sublist is considered "TRUE" if it has a positive element. Although you process the list step-by-step, you will only output the final list of booleans once all sublists have been evaluated.
Here are some examples to illustrate the task:
Example 1:
CONTEXT: [1, 1, 2, 3, -1, 2, 1]
#Answer: [True, True, True, True, True, True, True]
Example 2:
CONTEXT: [-0.1, -9, -0.11, 5, 0, 3.5]
#Answer: [False, False, False, True, True, True]
Example 3:
CONTEXT: [-1, -2, -3, -4, -5]
#Answer: [False, False, False, False, False]
Example 4:
CONTEXT: [-0.5, 1.5, -0.5, 2.5, -2.5]
#Answer: [False, True, True, True, True]
Example 5:
CONTEXT: [-10, -10, -3, 0.01, -0.01]
#Answer: [False, False, False, True, True]
Now, given a new list of numbers, perform the same task and provide the final output in the specified format.
IMPORTANT:

- DO NOT include any other text in your response.
- DO NOT use any PYTHON code in your response.
- GIVE JUST THE OUTPUT LIST AS THE ANSWER.

L Example of an output generated by a fine-tuned Llama 3.1 8B on Linear functions

Llama 3.1 8b fine-tuned on Linear Function did not understand the task and was not consistent even with the size of the output generated, which was different from an example to another. Here is an example of the generated output $[1.33, 0.79, 2.61, 0.0, 0.9]$.

M Error values for different models

To ensure that comparisons between models are meaningful, for each $\mathcal{N}(0, \sigma)$, we set a seed when generating the 100 random linear functions, ensuring that each model sees the same randomly chosen functions and the same set of prompting points x_i.

The heatmap on the right in Figure 11 shows how our models generalized outside of the training distributions $D_{\mathcal{F}}, D_{\mathcal{I}} \sim \mathcal{N}(0, 1)$. Shifts both in $D_{\mathcal{I}}^{test}$ from $D_{\mathcal{I}}$ and in $D_{\mathcal{F}}^{test}$ from $D_{\mathcal{F}}$ prompted performance to degrade as can be seen in Figure 11. For error values on various distributions and models see Table 6 in Appendix M.

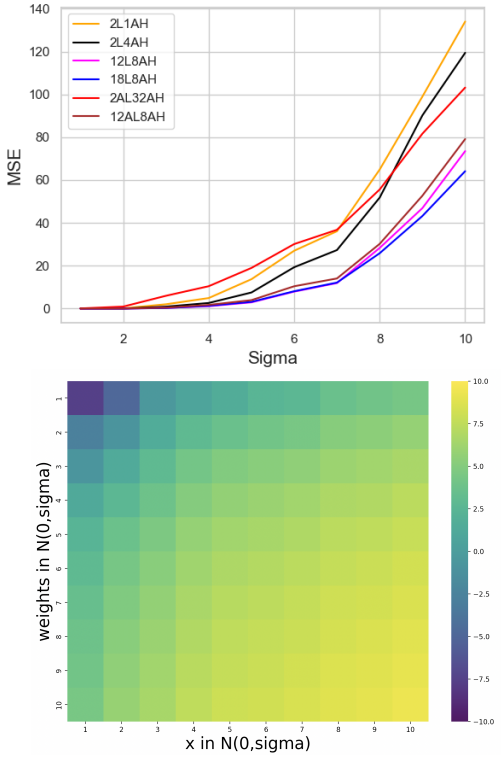

Figure 11: Evolution of MSE for various models with $D_{\mathcal{F}}, D_{\mathcal{I}}, D_{\mathcal{I}}^{test} \sim \mathcal{N}(0,1)$ and $D_{\mathcal{F}}^{test} \sim \mathcal{N}(0,\sigma)$ for various n. The heatmap shows error evolution when varying $D_{\mathcal{I}}^{test}$ and $D_{\mathcal{F}}^{test}$

.