SSA: Improving Performance With a Better Scoring Function

Omar Naim
IRIT
Université de Toulouse
omar.naim.docs@gmail.com

Swarnadeep Bhar
IRIT
Université de Toulouse
swarnadeep.bhar@irit.fr

Jérôme Bolte
Toulouse School of Economics
University of Toulouse Capitole
jbolte@ut-capitole.fr

Nicholas Asher
IRIT
CNRS
nicholas.asher@irit.fr

Abstract

While transformer models exhibit strong in-context learning (ICL) abilities, they often fail to generalize under simple distribution shifts. We analyze these failures and identify Softmax, the scoring function in the attention mechanism, as a contributing factor. We propose **Scaled Signed Averaging (SSA)**, a novel attention scoring function that mitigates these failures. SSA significantly improves performance on our ICL tasks and outperforms transformer models with Softmax on several NLP benchmarks and linguistic probing tasks, in both decoder-only and encoder-only architectures.

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; Naim et al., 2025a). 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 Transformer model (Nemotron-style, GPT-2–like) trained from scratch on the *FineWeb* corpus (Penedo et al., 2024) with SSA achieves lower perplexity and stronger performance than a Softmax-based counterpart across several standard NLP benchmarks. We also evaluate SSA in encoder architectures. In particular, we train several variants of BabyBERTa on a corpus of child-directed speech, including a standard Softmax model and SSA-based counterparts. Consistent with our decoder-only results, SSA-based BabyBERTa models achieve improved performance on multiple 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. (2024) 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 emerges through training, so studying it requires training from scratch. In this work, we investigate both types of 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 are matched, and Bhattamishra et al. (2023) showed ICL of Boolean functions in small GPT-2–style models. Raventós et al. (2023) analyzed how ICL capabilities evolve with pretraining scale under matched distributions. For quantification, Asher et al. (2023) proposed encoding semantic situations

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 30033–30051

as input sequences to assess a generative model's understanding of basic quantifiers. We adopt both of these evaluation paradigms: we test linear function prediction under distribution shift, and probe a model's grasp of "*every*" and "*some*" by encoding quantified situations in context and evaluating the correctness of the model's interpretations.

Limits of In-Context-Learning A growing body of work highlights sharp limits of ICL under distribution shift. Performance degrades substantially when inference-time inputs differ from the training distribution (Xie et al., 2021; Zhang et al., 2024; Giannou et al., 2024). Naim et al. (2025b) systematically characterized distribution shifts causing significant degradation, showing the failure is neither overfitting nor memorization. Ye et al. (2024); McCoy et al. (2024) demonstrate broader limits inherent to autoregressive training. Naim et al. (2025a) extend this analysis to provide theoretical characterizations of ICL failure modes. Our work complements these findings by identifying a concrete architectural mechanism, Softmax saturation, as a root cause, and by proposing a targeted fix.

Problems with Softmax Prior work has documented concentration effects inherent to the Softmax operation in 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 (Tian et al., 2023; Vig, 2019; Htut et al., 2019). Beyond token-level collapse, models may rely on only a few tokens, or even a single token per prompt, for prediction (Sekhsaria et al., 2025), with decision weight disproportionately concentrated on the final token (Mamidanna et al., 2025). Several alternative normalization functions have been proposed 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 entirely. Entmax (Peters et al., 2019) generalizes both Softmax and Sparsemax through a parametric family, enabling controllable sparsity between dense and sparse attention regimes. Temperature-scaled Softmax (Hinton et al., 2015) adjusts the entropy of the attention distribution via a scalar temperature parameter. More recent approaches include SA-Softmax (Zheng et al., 2025), which adapts logit scaling dynamically, and CosFormer (Qin et al., 2022), which replaces the exponential kernel with a cosine-modulated linear alternative. Despite their diversity, we show in Section 7 that none of these alternatives yields consistent improvements over standard Softmax on our tasks, motivating the design of SSA.

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 Setup

3.1 Tasks

We introduce two controlled ICL tasks and experimental setup used throughout the paper. The aim is to isolate the role of the attention scoring function and determine whether generalization failures arise from architectural choices rather than data complexity or scale.

The tasks probe complementary capabilities: logical aggregation and functional inference. Both tasks use synthetic data, operate on sequences of numbers and use transparent prompting to minimize confounds from prompt design.

Quantification task. This task evaluates whether a model can correctly interpret simple quantifiers over a sequence (see Figure 1). Given a sequence of numbers, the model must predict the truth of statements such as:

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

Training sequences S are drawn from an input distribution $D_{\mathcal{I}} = \mathcal{N}(0, 1)$.

At test time, we evaluate generalization under two shifts: (i) longer sequences S^{test} with lengths ranging from 10 to 200, and (ii) changes in input scale, with $D_{\mathcal{I}}^{\text{test}} = \mathcal{N}(0, \sigma)$ for $\sigma \geq 1$.

Linear function task. In this task, the model must infer an affine function $f(x) = ax + b$ from

in-context examples. Coefficients (a, b) are sampled from $D_{\mathcal{F}} = \mathcal{N}(0, 1)$, and inputs x_i are drawn independently from $D_{\mathcal{I}}$.

At test time, we vary both the input distribution $D_{\mathcal{I}}^{\text{test}} \sim \mathcal{N}(0, \sigma_1)$ and the function distribution $D_{\mathcal{F}}^{\text{test}} \sim \mathcal{N}(0, \sigma_2)$, with $\sigma_1, \sigma_2 \geq 1$.

3.2 Why these tasks?

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.

3.3 Training and Evaluation Setup

We train models from scratch on sequences of input-output pairs $(x_1, f(x_1), \ldots, x_i)$ followed by a query input x_i, for which the model must predict $f(x_i)$. Sequence lengths are sampled using a curriculum ranging from 11 to 40.[1]

We use decoder-only transformer models (GPT-2 style), ranging from 1 to 18 layers. Unless otherwise specified, we report results for a 12-layer model with 8 attention heads "12L8AH" (22.5M parameters), with and without MLP components, as larger models did not yield qualitatively different results.

To identify which components are responsible for ICL and its limitations, we performed ablation studies by systematically removing architectural components.

Training minimizes the expected autoregressive loss:

$$\mathop{\mathbb{E}}_{\substack{f \sim \mathcal{D}_f \\ x_1, \ldots, x_n \sim \mathcal{D}_I}} \left[\sum_{i=1}^{n} \ell\big(f(x_i), \mathcal{L}_\theta(x_1, f(x_1), \ldots, x_i)\big) \right]$$

where ℓ is squared error for the linear task and cross-entropy for the quantification task. Models are trained for 500,000 steps with batch size 64.

Evaluation. For the quantification task, we evaluate performance over pairs $(S^{\text{test}}, D_{\mathcal{I}}^{\text{test}})$ by generating 100 samples (each consisting of 64 batches) and reporting the average error rate. For the linear function task, we sample 100 functions from $D_{\mathcal{F}}^{\text{test}}$. For each function, we generate batches of input sequences drawn from $D_{\mathcal{I}}^{\text{test}}$ and evaluate predictions at each position k > 2 given the preceding context. We report mean squared error averaged across all prediction points, batches, and sampled functions.

This protocol systematically measures generalization under distribution shift across both tasks. The following sections use this setup to expose systematic generalization failures, analyze their origins, and ultimately identify the architectural component responsible.

4 ICL Results

In this section, we use the controlled setup introduced in Section 3.1 to evaluate whether models generalize under distribution shift across both tasks.

ICL with quantifiers. When test samples are drawn from the same distribution as training, i.e. $D_{\mathcal{I}}, D_{\mathcal{I}}^{\text{test}} \sim \mathcal{N}(0, 1)$, models successfully predict the correct truth values for (1-a) and (1-b), even for test sequences S^{test} substantially longer than those seen during training (Figure 3). However, performance drops sharply when inputs include one or more x_i values far outside the training distribution (Figure 1). We refer to such sequences as *deviant*.

ICL with linear functions. We replicate the findings of (Naim et al., 2025b): when training and test data are both sampled from $\mathcal{N}(0, 1)$, even small models achieve near-zero average error. All models exhibit systematic non-zero errors when the target function is drawn from a shifted distribution $D_{\mathcal{F}}^{\text{test}} = \mathcal{N}(0, \sigma)$ with $\sigma > 2$ (Appendix L).

Takeaway. Across both tasks, models perform well in-distribution but fail systematically under distribution shift, despite the underlying rule remaining

[1] The code is available at https://github.com/omyokun/SSA/.

unchanged. The following section analyzes the source of these failures.

5 Error Analysis

Having seen failures under distribution shift, we identify in this section what goes wrong when models encounter deviant sequences, and locate the failure within the architecture.

What goes wrong. Across both tasks, the failure pattern is the same: a single sufficiently large value dominates the model's output, causing it to ignore other elements crucial for correct prediction. In the quantification task, models base their predictions for an entire deviant sequence S almost exclusively on the largest element in S. The presence of a single such number is enough to trigger this behavior consistently (Figure 1). In the linear function task, a single out-of-range input value similarly disrupts predictions across the entire sequence (Figure 12 and Table 5).

Attention is the locus of failure. Crucially, this behavior appears in both attention-only and full transformer models across all our training and testing setups. As with (Olsson et al., 2022), ICL was effective even in models composed solely of attention layers, with no feedforward components (FF); these attention-only models performed comparably to their full transformer counterparts (Figure 13). In contrast, models consisting only of FF layers failed to perform ICL entirely. The attention mechanism is therefore both necessary and sufficient for ICL on our tasks. Since the same failure pattern appears with and without FF components, it cannot be attributed to the MLP but originates in the attention mechanism itself.

To rule out further representational issues, we verified that models could correctly classify individual numbers in deviant sequences as positive or negative (Figure 7). They performed well on this subtask, confirming that the information needed for correct prediction was available, but could not be used in the right way.

The failure is not a matter of scale. We observed similar issues with larger pre-trained models. We evaluated performance on the quantification task using both fine-tuned and prompted versions of LLaMA 3.1 8B, as well as the prompted LLaMA 3.3 70B model.[2] 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 handled large numbers within a sequence (Figure 11), but still did not generalize beyond sequence lengths seen in training. On the linear function task, prompted LLaMA 3.3 70B sometimes appeared to apply a regression-like strategy but still underperformed relative to our small models (Table 4).

Takeaway. The generalization failure is architectural, not a consequence of model scale or data representation. Since the attention mechanism is both necessary and sufficient for ICL, and the failure persists regardless of scale, the scoring function within attention becomes the prime suspect. We examine this in the next section.

6 The Softmax Problem

In this section, we show both mathematically and empirically why the Softmax scoring function is a primary cause of ICL generalization failures under distribution shift and the attention collapse observed in the previous section.

Softmax saturation. To recall the basics of attention, let $\mathrm{e}^{\ell} = (\mathrm{e}_1^{\ell}, \ldots, \mathrm{e}_n^{\ell})$ be the input embeddings processed by the multi-head attention mechanism at layer ℓ, where e_i^{ℓ} denotes the embedding of the i-th token. Each attention head h in layer ℓ is parameterized by Query, Key, and Value weight matrices $\mathrm{Q}^{h,\ell}$, $\mathrm{K}^{h,\ell}$, and $\mathrm{V}^{h,\ell}$, with d_k the embedding size divided by the number of heads. The output of each head is:

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

Once the gap between values in the Softmax argument exceeds a threshold, the distribution concentrates on its maximum. Let z_j denote the logits and assume $z_{j_0} = \max_j z_j$. Writing $\Delta_j = z_{j_0} - z_j \geq 0$, we have

$$\mathrm{softmax}(z_{j_0}) = \frac{1}{1 + \sum_{k \neq j_0} e^{-\Delta_k}}$$

$$\mathrm{softmax}(z_j) = \frac{e^{-\Delta_j}}{1 + \sum_{k \neq j_0} e^{-\Delta_k}}$$

[2] Prompts are provided in Appendix N.

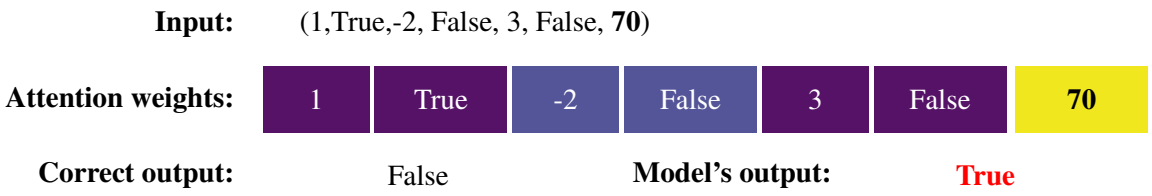

Figure 1: Attention maps for an ICL example for the task "every" of type $(x_1, f(x_1), x_2, f(x_2), ..., x_n)$, where the query x_n is a big value. Dark blue indicates weights approaching 0, while yellow indicates weights approaching 1.

If $\Delta_j \geq \delta$ for all $j \neq j_0$, then

$$\text{softmax}(z_{j_0}) \geq \frac{1}{1 + (n-1)e^{-\delta}}$$

$$\text{softmax}(z_j) \leq e^{-\Delta_j} \leq e^{-\delta}.$$

Thus, for $\delta \approx 4$ (so $e^{-4} \approx 0.018$), the maximal weight is close to 1 while all others are negligible, and the output collapses to:

$$\mathrm{C}_i^{h,\ell} = \mathrm{V}^{h,\ell}\mathrm{e}_{j_0}^{\ell} \quad (1)$$

making the attention head's representation of the token dependent only on a context of size one. Importantly, the large values x_j yielding Equation 1 come from inputs the model has seldom seen in training; the Q and K matrices were never trained to handle them.

Empirical confirmation. Following standard practice for ICL in small transformers, we use a linear embedding $\text{emb} : x \mapsto x \cdot W$, for $(x, W) \in \mathbb{R} \times \mathbb{R}^d$. This embedding preserves magnitude ordering: if $|x| < |y|$ then $\|\text{emb}(x)\| < \|\text{emb}(y)\|$. As a direct consequence, a large input value will always produce a large embedding norm, making Softmax saturation not merely possible but inevitable when deviant elements are present. Figure 1 confirms this prediction directly: when the sequence contains the value 70, the attention layer assigns virtually all weight to that token, ignoring the elements that actually determine the truth value of (1-a). The model consequently predicts the sequence as all positive, based solely on the large value. With significant differences in input values, Softmax increasingly resembles Hardmax, assigning weight close to 1 to the largest element and near 0 to all others. This concentration effect is not limited to linear embeddings: significant norm differences also arise with linguistic tokens, as seen in the *OpenWebText* corpus (Figure 9, Appendix H).

Effect on linear functions. Softmax saturation adversely affects the linear function task as well. For example, a 12L8AH model predicting $f(x) = x$ with input sequence $[100, -1.09, 0.78, 0.26, 0.42]$ produces predictions $[-1.21, -0.28, 2.15, 0.96, 0.65]$, a complete failure to approximate the function.

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. 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.

Takeaway. Softmax saturation is a fundamental architectural limitation: it causes attention to collapse onto a single token whenever large-magnitude inputs are present, regardless of whether that token is relevant to the task. This is particularly harmful for ICL, which requires integrating information across many context tokens simultaneously.

7 Exploring Alternatives to Softmax

Having identified Softmax saturation as the source of generalization failures, a natural question is whether existing alternative scoring functions already address this problem. In this section we answer that this is not the case.

We systematically evaluated a broad range of alternatives: temperature-scaled Softmax with $\tau \in \{5, 10, 20, 50, 100\}$ or a trainable temperature parameter; sparse attention mechanisms (Sparsemax (Martins and Astudillo, 2016) and Entmax (Peters et al., 2019)); hybrid schemes partitioning attention heads across tanh, uniform averaging,

ReLU, and x^2 scoring functions (details in Appendix C); and linear or modified Softmax approaches, namely linear attention, CosFormer (Qin et al., 2022) and SA-Softmax (Zheng et al., 2025). None of these yielded consistent improvements over standard Softmax on our tasks.

Takeaway. Despite the breadth of alternatives tested, none succeeded in mitigating the saturation problem identified in the previous section. This suggests that a more targeted solution is needed, which we develop in the next section.

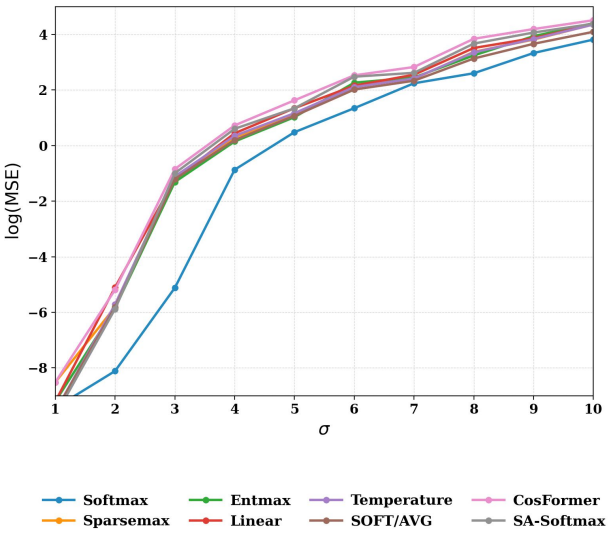

Figure 2: Evolution of $\log(\text{MSE})$ across models tested on $x \in D_{\mathcal{I}}^t = \mathcal{N}(0,1)$ and weights $a, b \in D_{\mathcal{F}}^t = \mathcal{N}(0,\sigma)$, as a function of the distribution shift parameter σ. All models use a 12-layer, 8-head full transformer. Exact values are reported in Table 4 in Appendix C.

8 Solution: Signed Scaled Averaging (SSA)

For ICL tasks that require integrating information across many tokens, Softmax saturation is clearly harmful. Conversely, tasks requiring focused attention on specific tokens can benefit from it. Rather than relying on a one-size-fits-all scoring function, we propose a mechanism that can adaptively control the degree of selectivity.

Selective Scaled Attention (SSA). To improve ICL performance, we replace the exponential scoring function with a parameterized alternative applied elementwise:

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

where $b > 0$ and $n \geq 1$ are trainable parameters.

The absolute value ensures a positive base, while the signed exponent induces asymmetric behavior: positive inputs grow polynomially, whereas negative inputs decay toward zero.

For a vector $z = (z_1, \ldots, z_p) \in \mathbb{R}^p$, SSA is defined as

$$\mathrm{SSA}(z)_i \;=\; \frac{(1 + b\,|z_i|)^{\operatorname{sgn}(z_i)\, n}}{\sum_{k=1}^{p} (1 + b\,|z_k|)^{\operatorname{sgn}(z_k)\, n}} \qquad (2)$$

Interpolation between regimes. SSA defines a continuous family interpolating between polynomial and exponential behaviors. For $x \geq 0$, setting $b = \frac{1}{m}$ and $n = m$ yields

$$(1 + bx)^n = \left(1 + \tfrac{x}{m}\right)^m \xrightarrow[m\to\infty]{} e^x,$$

recovering exponential growth. At the other extreme, for $x \geq 0$ and when $b = n = 1$, we obtain

$$(1 + bx)^n = 1 + x,$$

which is linear.

Contextualization: SSA vs Softmax SSA differs from Softmax under two key transformations of the logits. The first is global magnification: the logits are multiplied by a common factor, which can occur through the embedding or under out-of-distribution inputs in our ICL tasks. The second is inner disproportion: one token has an increasingly larger score than the others like in the experiments in Figure 1.

In both cases —magnification and inner disproportion— SSA tends to keep a more balanced view of context, whereas Softmax rapidly tends towards hardmax, even in the presence of temperature.

In this paragraph, b and $n \geq 1$ are fixed, so is the SSA map; similarly, we fix a temperature $\tau > 0$ for Softmax. Since non-positive logits rapidly vanish, we restrict to positive logits.

We first address the magnification question, where we see that Softmax saturates very fast while SSA keeps a more balanced view of very large logits.

Theorem 1 (Logit magnification) *Let $r \in \mathbb{R}^k$ be a positive vector and let $i_* = \arg\max\{r_i : i = 1, \ldots, k\}$ be a maximizing token, assumed to be unique. Given a scaling factor $\lambda > 0$, consider the two attention distributions*

$$\alpha_i^{\mathrm{SM},\tau}(\lambda) = \frac{e^{\lambda r_i/\tau}}{\sum_j e^{\lambda r_j/\tau}},$$

and

$$\alpha_i^{\mathrm{SSA}}(\lambda) = \frac{(1 + b\lambda r_i)^n}{\sum_j (1 + b\lambda r_j)^n},$$

for $i = 1, \ldots, k$. Then, as $\lambda \to \infty$,

$$\alpha_i^{\mathrm{SM},\tau}(\lambda) \to \begin{cases} 1, & i = i_*, \\ 0, & i \neq i_*, \end{cases}$$

while

$$\alpha_i^{\mathrm{SSA}}(\lambda) \to \frac{r_i^n}{\sum_j r_j^n}.$$

In other words, the residual "contextual mass" satisfies

$$1 - \alpha_{i_*}^{\mathrm{SM},\tau}(\lambda) \to 0,$$
$$1 - \alpha_{i_*}^{\mathrm{SSA}}(\lambda) \to 1 - \frac{r_{i_*}^n}{\sum_j r_j^n}, \quad \text{as } \lambda \to \infty.$$

Proof. For Softmax, divide all terms by $e^{\lambda r_{i_*}/\tau}$. If $j \neq i_*$, then

$$\frac{e^{\lambda r_j/\tau}}{e^{\lambda r_{i_*}/\tau}} = e^{-\lambda(r_{i_*} - r_j)/\tau} \to 0,$$

because $r_{i_*} > r_j$. Hence all nonmaximizing tokens vanish and $\alpha_{i_*}^{\mathrm{SM},\tau}(\lambda) \to 1$, while $\alpha_j^{\mathrm{SM},\tau}(\lambda) \to 0$ for $j \neq i_*$.

For SSA, since every r_i is positive,

$$(1 + b\lambda r_i)^n = (b\lambda)^n \left(r_i + \frac{1}{b\lambda}\right)^n.$$

The common factor $(b\lambda)^n$ cancels in the normalization. The remaining terms converge to r_i^n, giving the stated limit. □

We now study inner disproportion, understood here as the collapse of contextual mass when the imbalance between the relative importance or score of tokens increases.

Theorem 2 (Contextual mass collapse) *Let $s = (s_1, \ldots, s_k)$ in $\mathbb{R}^k$ be a positive vector and let i_* be the index of the maximizing token. Assume that $\rho := s_{i_*} \geq 1$ and $s_j \leq 1$ for all $j \neq i_*$. Then temperature-scaled Softmax satisfies*

$$1 - \alpha_{i_*}^{\mathrm{SM},\tau}(s) \leq (k-1)e^{-(\rho-1)/\tau},$$

whereas SSA satisfies

$$1 - \alpha_{i_*}^{\mathrm{SSA}}(s) = \frac{\sum_{j \neq i_*} (1 + bs_j)^n}{(1 + b\rho)^n + \sum_{j \neq i_*} (1 + bs_j)^n}$$

and so

$$1 - \alpha_{i_*}^{\mathrm{SSA}}(s) = O(\rho^{-n}).$$

Proof. For Softmax, since $s_j \leq 1$ for $j \neq i_*$,

$$1 - \alpha_{i_*}^{\mathrm{SM},\tau}(s) \leq \sum_{j \neq i_*} e^{-(\rho - s_j)/\tau}.$$

Since $s_j \leq 1$ for all $j \neq i_*$,

$$\sum_{j \neq i_*} e^{-(\rho - s_j)/\tau} \leq (k-1)e^{-(\rho-1)/\tau}.$$

For SSA, the displayed formula follows directly from the attention weights. Since $s_j \leq 1$ for $j \neq i_*$, the numerator is bounded by $(k-1)(1+b)^n$, giving the stated polynomial bound. □

In conclusion, as the size of the winning token comes to dominate, temperature-scaled Softmax tends to hardmax exponentially fast, while SSA moves only polynomially, allowing for a more subtle contextualization.

Additional theoretical analysis of SSA is provided in Appendix D.

Takeaway. Theorems 1 and 2 show, from two different perspectives, that SSA keeps a stronger awareness of contextual variety. Under global magnification, Softmax loses the contextual mass outside the winning token, whereas SSA preserves a distribution shaped by the relative scores. Under inner disproportion, Softmax moves to hardmax exponentially fast, whereas SSA collapses only polynomially. This matters for ICL, where the prediction often depends on several contextual examples rather than on a single dominant token. SSA therefore does not only soften attention: it gives each head a trainable way to delay collapse and keep secondary tokens active when the context still matters, while introducing only two scalar parameters.

9 Evaluating SSA

We now show how SSA's theoretical advantages translate into empirical gains, first on the controlled ICL tasks that motivated its design, then on broader NLP benchmarks to test whether the benefits generalize beyond synthetic settings.

9.1 Effects of SSA on ICL Tasks

Substituting SSA for Softmax substantially improves performance on both ICL tasks. All results use our 12L8AH full transformer model. For the quantification task, SSA yields considerable generalization improvement both in handling longer test sequences and in managing deviant inputs with extreme values. Figure 3 compares error rates

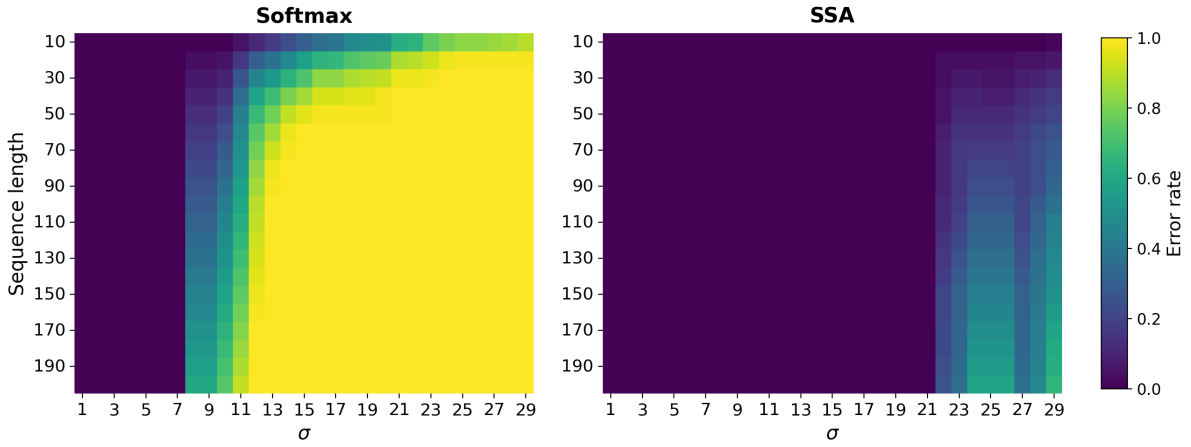

Figure 3: Heatmaps showing the evolution of errors for the 12L8AH model with Softmax (Left) and SSA (Right) on the *every* task. Model was trained on data in $D_{\mathcal{I}} = \mathcal{N}(0, 1)$ for lengths from 11 to 40 and tested in $D_{\mathcal{I}}^{test} = \mathcal{N}(0, \sigma)$ for $\sigma \in \{1, ..., 30\}$ and lengths from 10 to 200 for each task. Yellow represents a much higher error rate than purple.

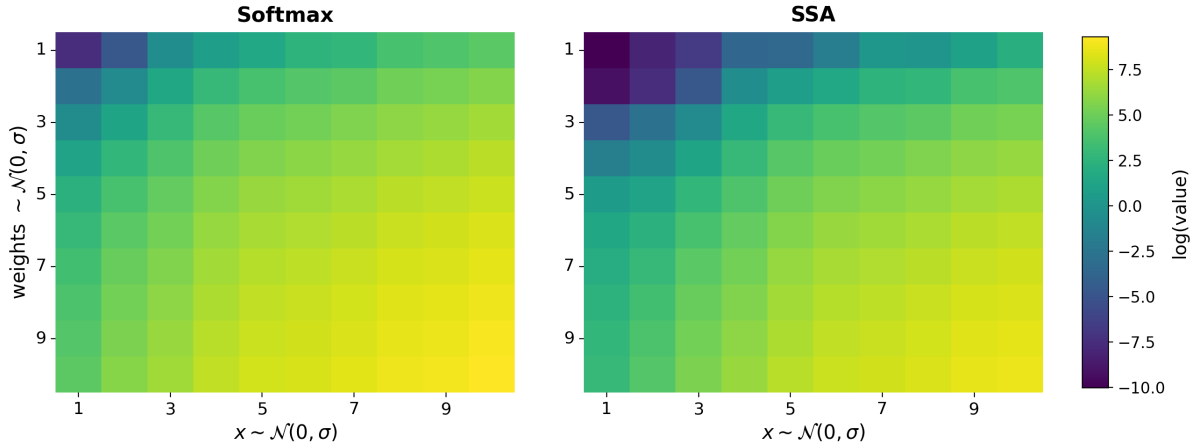

Figure 4: (Left) Comparison plot showing the evolution of log MSE for SSA and Softmax-based models (12L8AH) with $D_{\mathcal{F}}, D_{\mathcal{I}}, D_{\mathcal{I}}^{test} \sim \mathcal{N}(0, 1)$ and varying $D_{\mathcal{F}}^{test} \sim \mathcal{N}(0, \sigma)$. The heatmap shows the evolution of logarithm of MSE for the Softmax (Left) and SSA (Right) model when varying both $D_{\mathcal{I}}^{test}$ and $D_{\mathcal{F}}^{test}$.

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 performance on the *some* task (Appendix E). For the linear function task, SSA substantially improves performance when tested on out-of-distribution function parameters (Figure 4), outperforming all alternatives tested (Table 4).

Takeaway. SSA directly addresses the failure modes identified in Section 6: by avoiding saturation, it allows attention to remain distributed across context tokens even in the presence of large-magnitude inputs. We next ask whether this advantage persists in real NLP settings.

9.2 SSA on Decoder-Only NLP Benchmarks

We trained a Nemotron-style decoder-only model (114M parameters, 12 layers, 24 attention heads, hidden dimension 768) from scratch on 10B tokens from the FineWeb corpus for 22k steps , using a custom tokenizer with a vocabulary of 50,256 tokens. We compared a standard Softmax baseline against an SSA variant under identical training conditions, and evaluated both models on a range of benchmarks using the LM Eval harness (Biderman et al., 2024).

As shown in Table 1, SSA consistently outperforms Softmax across all evaluated tasks. Table 2 further shows that SSA achieves lower perplexity on both in-distribution FineWeb data and out-of-distribution Wikipedia text, with training loss

curves reported in Figure 8 (Appendix G).

Benchmark	Metric	Softmax	SSA
arc_challenge	acc_norm	0.2398 ± 0.0125	$\mathbf{0.2713 \pm 0.0130}$
arc_easy	acc_norm	0.2934 ± 0.0093	$\mathbf{0.5387 \pm 0.0102}$
boolq	acc	0.3783 ± 0.0085	$\mathbf{0.5618 \pm 0.0087}$
cb	acc	0.1429 ± 0.0472	$\mathbf{0.4643 \pm 0.0672}$
cb	f1	0.1310	**0.2663**
copa	acc	0.5900 ± 0.0494	$\mathbf{0.6400 \pm 0.0482}$
hellaswag	acc_norm	0.2550 ± 0.0043	$\mathbf{0.3283 \pm 0.0047}$
multirc	acc	0.4280 ± 0.0071	$\mathbf{0.4350 \pm 0.0071}$
openbookqa	acc_norm	0.2860 ± 0.0202	$\mathbf{0.3040 \pm 0.0206}$
record	f1	0.1983 ± 0.0040	$\mathbf{0.2482 \pm 0.0043}$
record	em	0.1932 ± 0.0039	$\mathbf{0.2427 \pm 0.0043}$
wic	acc	0.5000 ± 0.0198	$\mathbf{0.5078 \pm 0.0198}$
winogrande	acc	0.4972 ± 0.0141	$\mathbf{0.5178 \pm 0.0140}$

Table 1: Zero-shot benchmark comparison between Softmax and SSA on a Nemotron-style decoder model (114M) trained for 22K steps on FineWeb. Bold indicates the better result.

Evaluation	Softmax	SSA
FineWeb (in-dist.) ↓	21.86	**19.73**
Wikipedia (out-dist.) ↓	24.58	**22.07**

Table 2: Perplexity comparison between Softmax and SSA (114M Nemotron-style decoder, 22K steps). Bold indicates the better result.

Takeaway. SSA's benefits are not confined to synthetic tasks: it achieves lower perplexity on both in-distribution and out-of-distribution text, lower training loss, and consistent gains across a diverse set of NLP benchmarks, all after only 22k steps of training on FineWeb.

9.3 SSA on Encoder-Only Models

To test SSA beyond decoder-only architectures, 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, well-suited for probing grammatical dependency learning in limited-data settings. We evaluated the models on 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.

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.

Takeaway. Consistent with the decoder-only results, SSA enhances grammatical sensitivity in encoder-only models as well. Taken together, the results across controlled tasks, decoder models, and encoder models support SSA as an effective drop-in replacement for Softmax across transformer architectures.

10 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 114M parameters. We trained a Nemotron-style decoder-only model

(114M parameters, 12 layers, 24 attention heads, hidden dimension 768) from scratch on 10B tokens from the FineWeb corpus for 22k steps, using a custom tokenizer with a vocabulary of 50,256 tokens. This required approximately 2 days on 12 NVIDIA A100 80GB GPUs per model. To enable a rigorous comparison, we trained both a Softmax and an SSA version under identical conditions, resulting in a total of over 4 days of compute.

While SSA provides consistent improvements in generalization, it does not fully resolve all limitations of the attention mechanism. In particular, our results indicate that SSA still struggles in scenarios involving strong distribution shifts, where both the input x_i and the test-time function distribution $D_{\mathcal{F}}^{\text{test}}$ differ significantly from the training distribution $D_{\mathcal{F}}$.

More fundamentally, this points to a broader open question regarding the attention mechanism itself: its underlying mathematical structure tends to conflate the magnitude of token representations with their relevance to the task. As a result, tokens with large values can disproportionately influence attention, even when they are not the most informative for the prediction. Addressing this misalignment remains an important direction for future work.

Acknowledgments

This work was supported by SARER and SummRE projects (ANR-20-CE23-0017), and the AI Cluster ANITI (ANR-19-PI3A-0004). Computational resources were provided by CALMIP under Grant 2016-P23060.

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, and 1 others. 2024. Lessons from the trenches on reproducible evaluation of language models. *arXiv preprint arXiv:2405.14782*.

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.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, and 1 others. 2024. A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 1107–1128.

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*.

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, Liang 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.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 30735–30748.

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, Jérôme Bolte, and Nicholas Asher. 2025a. Analyzing limits for in-context learning. *arXiv preprint arXiv:2502.03503*.

Omar Naim, Guilhem Fouilhé, and Nicholas Asher. 2025b. Re-examining learning linear functions in context. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 104–117. Springer.

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*.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin 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. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1504–1519.

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. *arXiv preprint arXiv:2202.08791*.

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. 2023. Pretraining task diversity and the emergence of non-bayesian in-context learning for regression. *Advances in neural information processing systems*, 36:14228–14246.

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*.

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*.

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, Zhenguo Li, and Yu Li. 2025. Self-adjust softmax. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7827–7847.

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 (Kingma and Ba, 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 (Kingma and Ba, 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 $(w_1, w_2, ...)$. It failed to capture the basic input and output pattern.

A.2 Training on FineWeb

Training Details. We trained a Nemotron3-style Transformer decoder model from scratch with customized dimensions, totaling 114M parameters. The model consists of 12 Transformer layers with 24 attention heads and grouped-query attention (GQA) with 8 query groups, a hidden dimension of 768, a feed-forward hidden dimension of 3,072, and a maximum context length of 1,024 tokens. Input and output embeddings are shared, and the vocabulary size is 50,256 tokens. Training was performed using the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\varepsilon = 10^{-5}$, and a weight decay of 0.1. We used a peak learning rate of 3×10^{-4} with a cosine decay schedule over the full training horizon, computed from the total token budget of approximately 10B tokens. The global batch size was 128 sequences with a micro-batch size of 1 and a sequence length of 1,024 tokens, yielding approximately 131k tokens per step. Mixed-precision training was conducted in BFloat16 using a distributed optimizer. For SSA, the parameter b was fixed to 0.8, while n was learned as a per-layer parameter (shared across attention heads).

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 8 attention heads per layer, this corresponds to a total of $2 \times 12 \times 8 = 192$ additional parameters, which is negligible compared to the overall model size. All 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.

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,

models \ σ	1	2	3	4	5	6	7	8	9	10
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
Linear	10^{-4}	6.1×10^{-3}	0.3	1.56	3.84	8.64	12.82	33.54	47.86	79.06
Trainable τ	7×10^{-5}	3.3×10^{-3}	0.33	1.42	3.21	8.21	11.22	28.60	46.22	78.02
$\tau = 5$	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 4: Comparison showing the evolution of squared errors for models tested on $x \in D_{\mathcal{I}}^t = \mathcal{N}(0,1)$ and weights $a, b \in D_{\mathcal{F}}^t = \mathcal{N}(0,\sigma)$. Temperature-scaled Softmax is evaluated with either a fixed temperature $\tau \in \{5, 10, 20, 50, 100\}$ or a trainable temperature parameter learned during training. All models use a 12-layer Transformer with 8 attention heads.

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 4). 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 4).

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 4).

D Additional Supporting Analysis for the SSA Function

Let's consider:

$$f(x) = (1 + b|x|)^{\operatorname{sgn}(x)\, n}, \qquad b > 0,\ n \geq 1, \tag{3}$$

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

$$\alpha_i^{\text{SSA}} = \frac{f(s_i)}{\sum_{j=1}^{k} f(s_j)}. \tag{4}$$

In this section, we provide additional analytical details for the function f, including its C^1-regularity.

D.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.

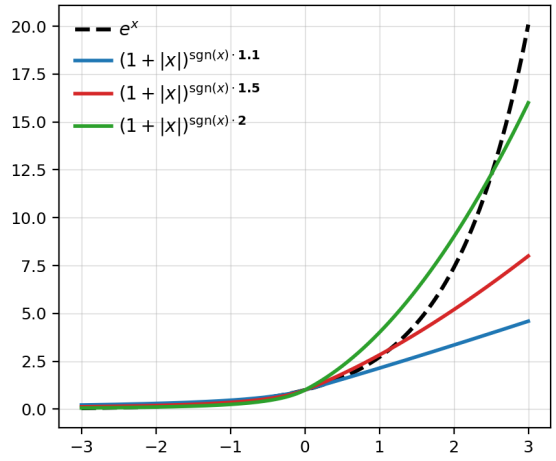

Figure 5: Illustration of the base function $(1 + b|x|)^{\text{sgn}(x)n}$ used in SSA, plotted for $b = 1$ and $n \in \{1.1, 1.5, 2\}$. The curves demonstrate that SSA exhibits behavior similar to the exponential function, but with a tunable growth rate toward $+\infty$, which increases with larger values of the exponent n.

D.2 Gradient Structure and Logarithmic Slope

Piecewise derivative. Differentiation yields

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

Therefore f is C^1 and

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

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

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

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

The graph of g(x) for ssa shows that extremely large inputs are "cropped"; an increase in the size of large input will only provide a small increase in score, whereas for small values SSA is quite sensitive. This does not occur with Softmax with $g(x) = 1$: large inputs give large outputs.

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

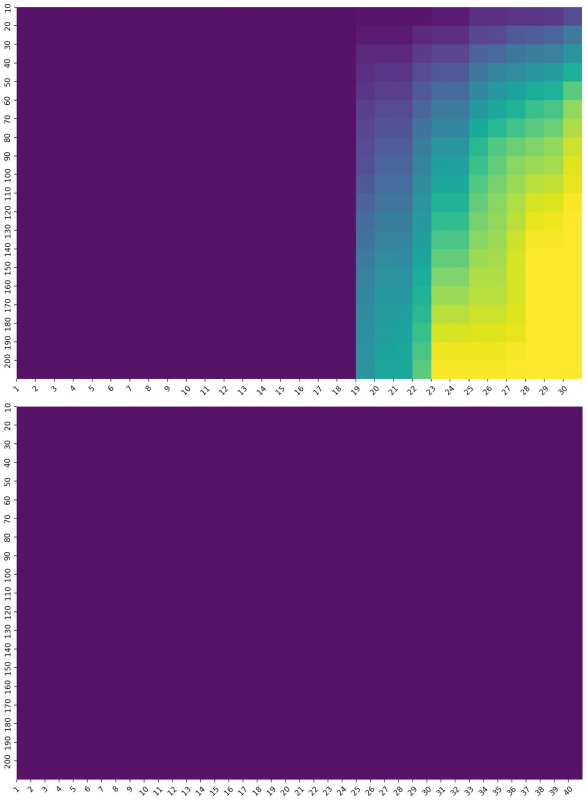

Figure 6: Heatmap showing the evolution of errors for the task *some* trained on data in $D_{\mathcal{I}} = \mathcal{N}(0, 1)$ for lengths from 11 to 40 and tested in $D_{\mathcal{I}}^{\text{test}} = \mathcal{N}(0, \sigma)$ for $\sigma \in \{1, ..., 30\}$ and lengths from 10 to 200. The first figure is for the Softmax-based model and the second with SSA. Yellow represents a much higher error rate than purple.

F Individual Sign Classification Under Distribution Shift

To confirm that generalization failures on the quantification task are not caused by the model losing the ability to represent or classify individual values, we tested whether the model can correctly determine the sign of a single out-of-distribution input. For each $\sigma \in \{1, \ldots, 29\}$, we presented the model with a one-point prompt where the input was drawn from $\mathcal{N}(0, \sigma)$ and the ground truth label was the sign of that input.

As shown in Figure 7, the model achieves perfect accuracy (1.0) across all values of σ, with zero errors out of 64 sequences in every condition, regardless of how far the input lies outside the training distribution $\mathcal{N}(0, 1)$.

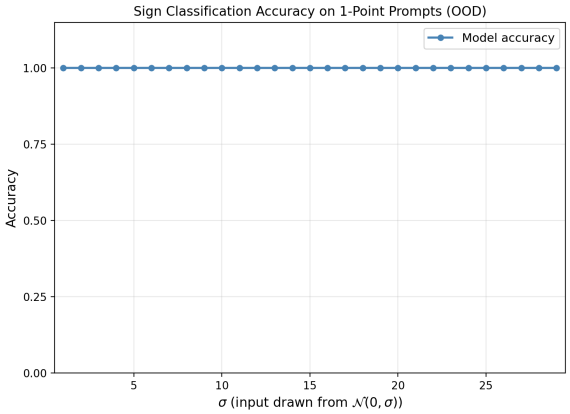

Figure 7: Sign classification accuracy of the 12L8AH Softmax model on one-point prompts with inputs drawn from $\mathcal{N}(0, \sigma)$, for $\sigma \in \{1, \ldots, 29\}$.

G Training Loss Evolution: SSA vs Softmax

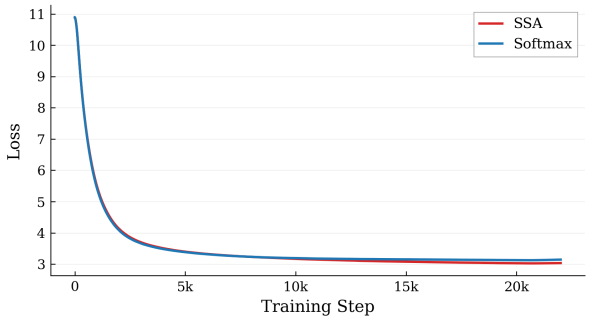

Figure 8: Training loss curves for SSA and Softmax on the Nemotron-style decoder model (114M) trained on FineWeb for 22K steps.

H Repartition of token norms in the OpenWebText

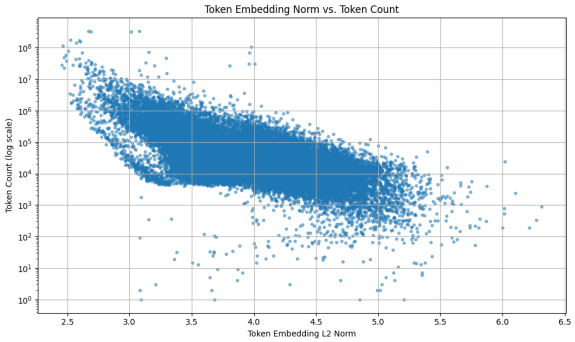

Figure 9: Plot showing how many tokens have a norm of value x, for OpenWebText using GPT-2 Tokenizer. The x-axis is token embedding norm and the y-axis is token count (log scale)

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

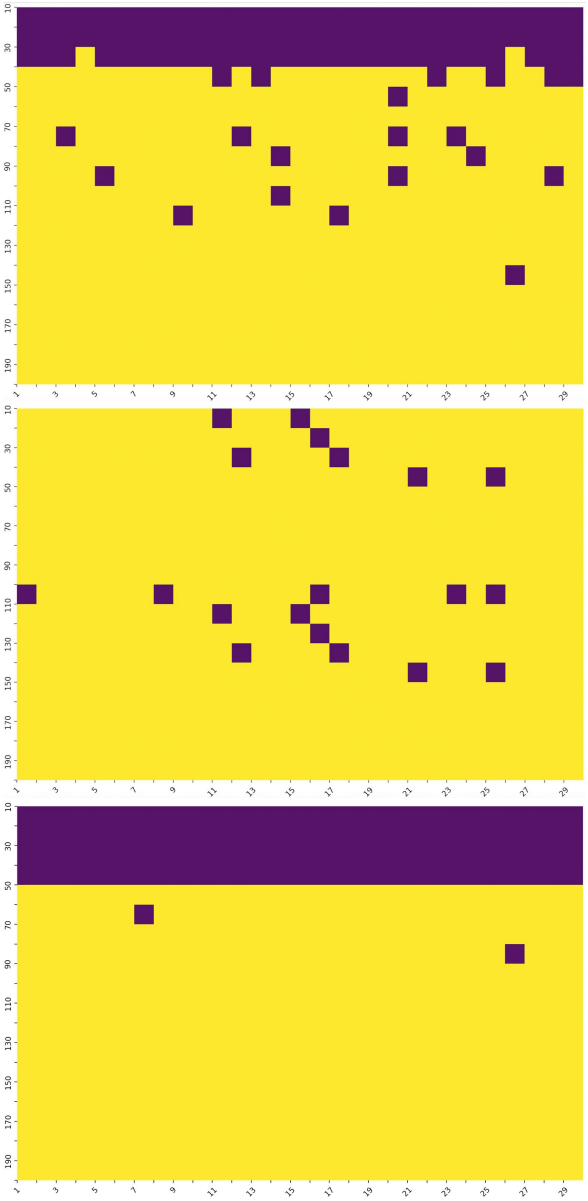

Figure 10: Heatmap showing the evolution of errors for the task "some" on different models "Pre-trained Llama 3.3 70B", "Pre-trained Llama 3.1 8B" and finetuned on Llama 31 8B", respectively from top to bottom on data in $D_{\mathcal{I}} = \mathcal{N}(0, 1)$ for lengths from 11 to 40 and tested in $D^t_{\mathcal{I}}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

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

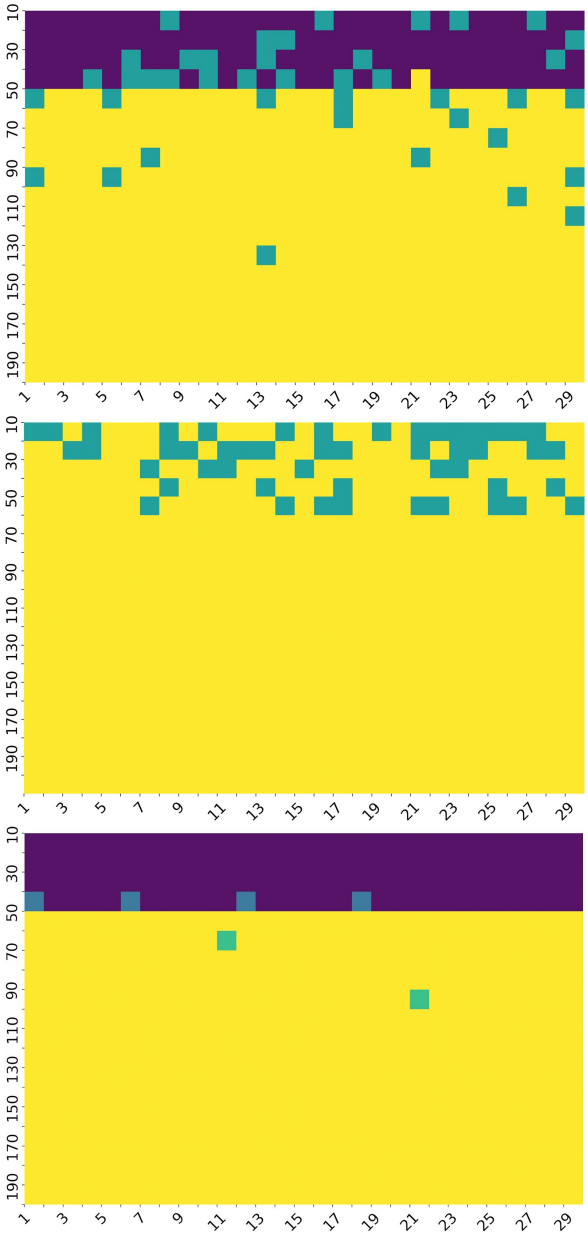

Figure 11: Heatmap showing the evolution of errors for the task "every" on "Pre-trained Llama 3.3 70B" (First), "Pre-trained Llama 3.1 8B" (Second) with 5 shot learning and finetuned on Llama 31 8B" (Third), on data in $D_{\mathcal{I}} = \mathcal{N}(0, 1)$ for lengths from 1 to 40 and tested in $D^t_{\mathcal{I}} = \mathcal{N}(0, \sigma)$ for $\sigma \in \{1, ..., 10\}$ and lengths from 10 to 200. Yellow represents a much higher error rate than purple.

K Examples of How Models Behave In-Distribution vs. Under Distribution Shift

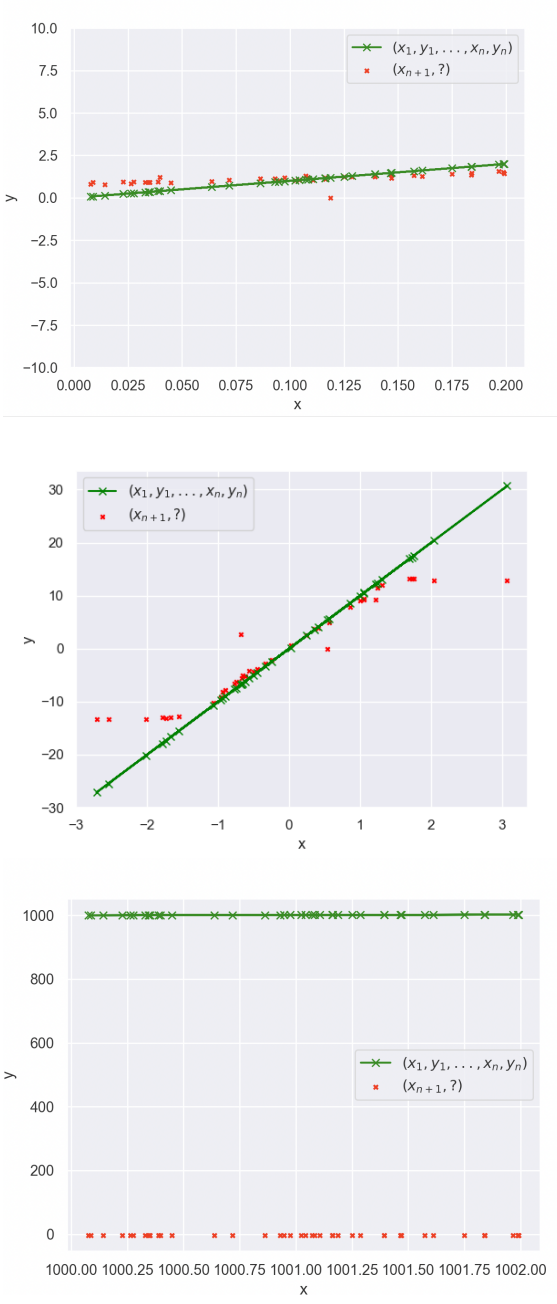

Figure 12: Plots for model 12L8AH, trained on $D_{\mathcal{I}}, D_{\mathcal{F}} \sim \mathcal{N}(0, 1)$ for $f(x) = x$ for high values (First) of x and $f(x) = 10x$ for normal (Second) then for low values of x (Third)

L Out-of-Distribution Generalization Failure Across Architectures

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.

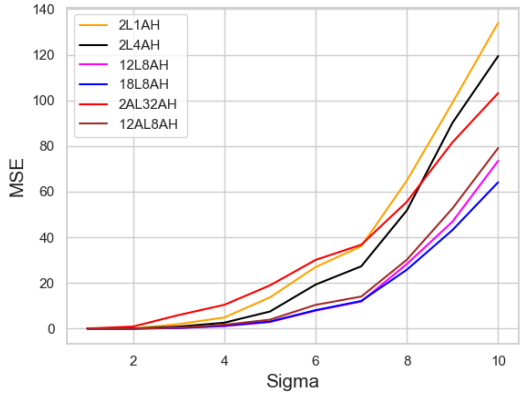

Figure 13: Evolution of MSE for full transformer models (e.g., 12L8AH = 12 layers, 8 heads) and attention-only models (e.g., 12AL8AH = 12 attention layers, 8 heads), with $D_{\mathcal{F}}, D_{\mathcal{I}}, D_{\mathcal{I}}^{test} \sim \mathcal{N}(0, 1)$ and $D_{\mathcal{F}}^{test} \sim \mathcal{N}(0, \sigma)$ for various σ. Both architectures exhibit strong in-distribution ICL performance ($\sigma \approx 1$), but MSE degrades sharply for all models as σ increases, revealing a consistent failure to generalize out-of-distribution regardless of architecture.

M Evolution of MSE over different scales of small Transformers

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 $a, b \in D_{\mathcal{F}}^t = \mathcal{N}(0,1)$.

N 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.

O 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 $[1.33, 0.79, 2.61, 0.0, 0.9]$.