REASONING DEEPER THROUGH CONTROLLED SHARPNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-attention in Transformers uses softmax to normalize query–key products into a distribution of attention scores. A sharp distribution concentrates attention on a small number of tokens, whereas a flat one spreads it more evenly across tokens. Softmax, however, has no parameter that directly controls this sharpness. Instead, sharpness depends indirectly on the magnitudes of the query–key products, emerging through training and varying markedly across random seeds. We identify an empirical association between attention sharpness and reasoning depth under softmax. Motivated by this association, we introduce powered L1 attention (L1p), a normalizing function whose sharpness is controlled by a learnable per-layer exponent and whose attention scores are invariant to the overall magnitude of the query–key products. We further introduce a signed form of L1p that permits negative attention scores, which our ablations show is critical for reasoning depth. On a synthetic multi-hop reasoning task with an exactly controllable number of hops, L1p reaches greater reasoning depth than softmax and five established alternative normalizing functions within the same training budget. At 4 hops, L1p solves the task in 18 of 32 runs with different random seeds, whereas every alternative succeeds in at most 3 of 32. At the same time, across 24 random seeds, L1p matches the language modeling perplexity of softmax within a pre-specified margin. These results show that directly controlling attention sharpness through an exponent, together with signed attention scores, can substantially increase the reasoning depth a Transformer achieves within a single forward pass.

1 INTRODUCTION

Autoregressive language models have become central to a wide range of applications, including conversational assistants (Ouyang et al., 2022), question answering (Brown et al., 2020), and code generation (Li et al., 2022), among others. Within these applications, some tasks require a model to draw information from its input in a single step. Others require several sequential steps, each depending on the result of the preceding one. Suppose, for instance, that the input states that Alice's manager is Bob and, elsewhere, that Bob sits in room 12. Answering where Alice's manager sits requires the model to first establish that the manager is Bob and only then determine where Bob sits. The second step depends on the result of the first. Questions of this kind are known as *multi-hop* questions (Welbl et al., 2018; Yang et al., 2018), with each step constituting a *hop*. We refer to the number of hops a model can successfully chain within a single forward pass as its *reasoning depth*.

The Transformer (Vaswani et al., 2017) underlies most modern autoregressive language models, with self-attention as its central mechanism. Such models are trained to predict the next token in a sequence. To support that prediction, self-attention computes a scaled dot product between a query vector and the key vector of each accessible token, producing a score that reflects the relevance of that token to the query. These scores are then normalized and used as weights in a weighted sum of the corresponding value vectors. The resulting representation is subsequently processed by the rest of the model to produce the next-token prediction. We denote the normalized scores as the *attention scores* and the function that produces them as the *normalizing function*.

For each hop of a multi-hop question, the Transformer should assign high attention scores to the tokens carrying the information needed for that hop, and lower scores to less relevant ones. An attention-score distribution in which a small number of tokens receive substantially larger scores than the rest is called *sharp*, whereas one in which scores are more evenly distributed across tokens

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

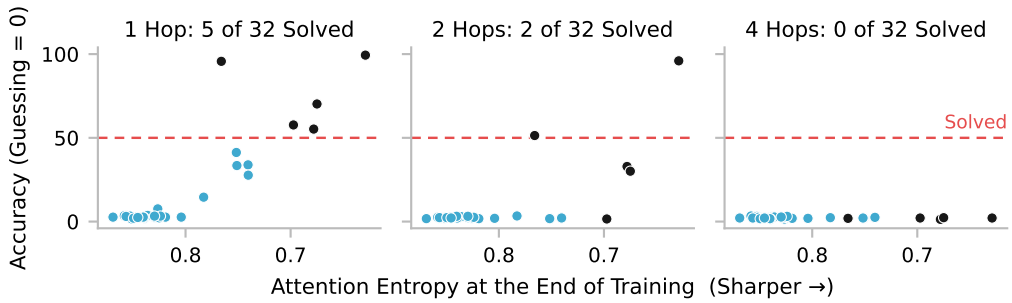

Figure 1: Final accuracy versus end-of-training attention sharpness for 32 runs of the same softmax Transformer with different random seeds, on the multi-hop reasoning task adapted from Allen-Zhu (2025). Each point is one run and each panel one hop count. Accuracy is rescaled from guessing (0) to perfect (100), with 50 marking a solved hop count. Sharpness is measured by entropy, with lower values indicating sharper attention. Black points mark the 5 runs that solve 1 hop.

is called *flat*. Because attention produces a weighted sum of the value vectors, a flatter distribution can mix information from relevant tokens with information from less relevant ones, making the computation at that hop less reliable. A sharper distribution, by contrast, can more strongly isolate the information needed for the hop.

The effect of this difference compounds as reasoning extends across multiple hops. In particular, the information recovered at one hop is needed to support the next, so an error at any stage can propagate through the remaining reasoning chain, while answering the overall question correctly requires *every* hop to succeed. Improving the reliability of individual hops can therefore increase the number of hops that a model can successfully chain, that is, its reasoning depth. This suggests a connection between the sharpness of attention across successive hops and the depth of reasoning the Transformer can sustain.

The normalizing function is the component of the attention mechanism that shapes the attention-score distribution. In the Transformer, this function is softmax. Softmax, however, has no parameters of its own that directly control sharpness. Instead, it operates on the query–key products, and when their *overall magnitude* increases (i.e., when they are scaled by a common positive factor while preserving their relative ordering), the resulting attention distribution becomes sharper. The sharpness of softmax attention is thereby governed indirectly by the overall magnitude of the query–key products. This magnitude, in turn, depends on the model's learned parameters, which are shaped by initialization and optimization. As a result, attention sharpness emerges from training rather than being directly controlled by the architecture. Empirically, we find that it varies markedly across training runs with different random seeds.

To illustrate this behavior, Fig. 1 shows 32 runs of the same Transformer model on a synthetic multi-hop reasoning task, each trained with a different random seed. Adapted from Allen-Zhu (2025), the task tests multi-hop traversal over a directed graph. For each run, we plot its final accuracy at 1, 2, and 4 hops against the sharpness of its attention-score distributions at the end of training. Only 5 of the 32 runs solve the task at 1 hop, only 2 solve it at 2 hops, and none solves it at 4 hops. Notably, as shown in the figure, the runs achieving higher accuracy at 1 and 2 hops tend to be those whose attention-score distributions are sharper. This empirical pattern reinforces the connection between attention sharpness and reasoning depth, while also exposing substantial variability across training runs of the softmax Transformer.

Collectively, these observations raise a natural question. What if the normalizing function instead had a dedicated parameter that directly controls its sharpness, while making that sharpness independent of the overall magnitude of its inputs and more consistent across random seeds? Could controlling attention sharpness by design enable a Transformer to reason more reliably and, crucially, more deeply? This is the central question we address in this paper.

Prior work characterized or extended reasoning depth through the number of Transformer layers, adaptive layer use, horizontal information flow, and cross-head communication (Sanford et al., 2024; Curth et al., 2026; Allen-Zhu, 2025; Duvvuri et al., 2026), but none varied the normalizing function on its own. A separate line of work replaced softmax for objectives such as sparse and interpretable attention, greater expressiveness, inference under homomorphic encryption, and robustness to distribution shift (Martins & Astudillo, 2016; Lv et al., 2024; Zimerman et al., 2026; Naim et al., 2026),

but did not investigate how the choice of normalizing function affects reasoning depth. Most closely related, Hoffmann et al. (2024) linked prolonged training plateaus in fixed two-step tasks to locally near-uniform softmax attention, but studied optimization in this setting rather than reasoning depth. To the best of our knowledge, the effect of a normalizing function with controlled sharpness on reasoning depth has not yet been studied.

To study this effect and directly address the research question posed above, we introduce **powered L1 attention** (L1p), a normalizing function whose sharpness is controlled by a learnable parameter in each layer. Given query–key products $z_1, \ldots, z_n$, L1p computes the attention scores as $|z_j|^p / \sum_l |z_l|^p$, where the exponent p is the learnable parameter. Increasing p concentrates more of the attention on inputs with larger magnitudes, whereas decreasing p distributes it more evenly. L1p is also *magnitude-invariant*, where multiplying every input by a constant $c > 0$ multiplies both the numerator and denominator by c^p, thus leaving the attention scores unchanged. As a consequence, the sharpness of L1p is governed by p rather than by the overall magnitude of the query–key products, unlike that of softmax.

We evaluate the resulting L1p against softmax, a signed variant of softmax, and five established alternative normalizing functions, namely, Cog Attention (Lv et al., 2024), ℓ_1 normalization (Tian et al., 2025), sparsemax (Martins & Astudillo, 2016), α-entmax (Peters et al., 2019), and Scaled Signed Averaging (Naim et al., 2026), on multi-hop reasoning and language modeling. The multi-hop reasoning task we use allows the number of hops to be controlled exactly, so evaluating accuracy across hop counts provides a direct measure of reasoning depth. All evaluations must also account for variation across random seeds. Fig. 1 illustrates this variability in the reasoning task, with only 5 of 32 runs solving the 1-hop task. A single run can thus confound the effect of the normalizing function with seed-dependent optimization. Our main comparisons accordingly use 32 seeds per normalizing function on multi-hop reasoning and up to 24 on language modeling. To the best of our knowledge, no prior study of multi-hop reasoning or comparison of normalizing functions has evaluated competing models across dozens of random seeds.

In summary, our main contributions are as follows:

- Under softmax, we identify an empirical association between attention sharpness and reasoning depth, and show that sharpness depends on the overall magnitude of the query–key products rather than being controlled by the architecture.
- We introduce L1p, a magnitude-invariant normalizing function with learnable per-layer sharpness and a signed form that permits negative attention scores. We compare it against softmax, a signed variant of softmax, and five established alternative normalizing functions.
- We evaluate each normalizing function with 32 seeds on multi-hop reasoning and up to 24 on language modeling. We further use 8 to 24 seeds per ablation and 16 per layer-count configuration. For every comparison supporting our conclusions, we report uncertainty and identify the statistical test, statistic, and sidedness.
- On multi-hop reasoning, L1p solves 4 hops in $18/32$ runs versus at most $3/32$ for any alternative; on language modeling, it matches softmax within the pre-specified 0.20 perplexity margin (Schuirmann, 1987).
- Ablations show that L1p's reasoning advantage depends on the value of its exponent and the placement of signed layers, and grows over softmax with the number of layers. From 4 to 20 layers, successful 4-hop L1p runs rise from $0/16$ to $15/16$, while softmax solves none at any tested layer count.
- We release our training and evaluation code to support reproducibility and future research.

2 Related Work

Table 1 compares L1p with a few representative prior normalizing functions, while Appendix G provides a comprehensive survey.

3 Background: Sharpness under Softmax

Self-attention consists of multiple heads, each independently computing an output from learned query, key, and value projections. Consider one head at the final position n of an autoregressive sequence of n tokens, where the head can attend to positions $1, \ldots, n$. Let $\mathbf{q}_n$ be its query vector and $\mathbf{k}_j$ and $\mathbf{v}_j$ the key and value vectors at position j. The scaled query–key product for position j is

Table 1: Properties of softmax and alternative normalizing functions. *Signed* means attention scores may be negative, while *magnitude-invariant* means they are unchanged under a common positive rescaling of the query–key products. The last two columns indicate whether a power above one is used and whether it can be learned.

	Signed	**Magnitude-Invariant**	$p > 1$	p **Learnable**
Softmax (2017)	✗	✗	✗	✗
Sparsemax (2016)	✗	✗	✗	✗
α-entmax (2019; 2019)	✗	✗	✓	✓
ℓ_1 normalization (2025)	✓	✓	✗	✗
Cog Attention (2024)	✓	✗	✗	✗
Scaled Signed Averaging (2026)	✗	✗	✓	✓
L1p (ours)	✓	✓	✓	✓

$z_j = \mathbf{q}_n^\top \mathbf{k}_j / \sqrt{d}$, where d is the query and key dimension. We refer to z_j as the *query–key product*, with the scaling by $\sqrt{d}$ assumed throughout. The vector $\mathbf{z} = (z_1, \ldots, z_n)$ forms the final row of the causal query–key matrix. Everything that follows applies to any row, with n denoting the number of positions it attends to.

A normalizing function maps $\mathbf{z}$ to attention scores $\mathbf{a}(\mathbf{z}) = (a_1, \ldots, a_n)$, which weight the value vectors in the head output $\sum_{j=1}^{n} a_j \mathbf{v}_j$. The standard Transformer uses softmax, with $a_j = \exp(z_j) / \sum_{l=1}^{n} \exp(z_l)$. A row is *sharp* when a small number of tokens receive substantially larger scores than the rest, and *flat* when scores are more evenly distributed. The ratio of two scores measures their relative weighting and, under softmax, satisfies $a_j / a_i = \exp(z_j - z_i)$.

Multiplying every product in $\mathbf{z}$ by $c > 1$ changes this ratio to $\exp(c(z_j - z_i))$, equivalently raising the original ratio to the power c. Any ratio greater than 1 accordingly increases, making favored tokens more strongly weighted and the row sharper. We call the mean magnitude of the entries of $\mathbf{z}$ their *overall magnitude*. Such scaling increases the overall magnitude while preserving the magnitudes relative to one another. Hence, softmax sharpness changes with the overall magnitude of its inputs.

This magnitude depends on the query and key vectors and can change during training for reasons unrelated to the sharpness required by the task. Wortsman et al. (2024) observed growth in this magnitude at high learning rates when studying large-Transformer training instabilities through small-scale proxies, while Dehghani et al. (2023) reported that this growth destabilized training of an 8-billion-parameter vision Transformer. Minimizing the training loss can thus enlarge the query–key products and sharpen softmax attention, whereas weight decay pulls parameters toward zero and opposes this growth. Softmax consequently leaves sharpness to these competing pressures rather than controlling it through a parameter of its own.

A learnable temperature provides such a parameter and was introduced in self-attention by Dufter et al. (2020) to improve learning efficiency. A scalar $\tau > 0$ divides every query–key product before softmax, yielding $a_j / a_i = \exp((z_j - z_i)/\tau)$. Smaller τ sharpens the row and larger τ flattens it. Temperature, however, does not remove the dependence on overall magnitude. Multiplying every product by $c > 1$ still raises this ratio to the power c, regardless of τ; equivalently, dividing τ by c has the same effect as multiplying every product by c. Changes in temperature can therefore be counteracted during training by changes in product magnitude.

In summary, softmax has two limitations for controlling sharpness. First, its sharpness changes with the overall magnitude of the query–key products. Second, it has no parameter of its own that controls sharpness. A learnable temperature provides such a parameter, but its effect remains coupled to the overall magnitude of the products. To this end, controlling sharpness calls for a dedicated parameter whose effect is invariant to the overall magnitude of the query–key products.

4 Powered L1 Attention

4.1 L1p

L1p is a normalizing function that addresses both limitations identified in Section 3. Its attention scores are invariant to the overall magnitude of the query–key products, and its sharpness is con-

trolled by a learnable parameter. We define L1p in two forms. The first, which we call the *unsigned* form, is

$$a_j = \frac{|z_j|^p}{\sum_{l=1}^{n} |z_l|^p}, \tag{1}$$

where p is a learnable scalar that we refer to as the *exponent*. At $p = 1$, the unsigned form divides the absolute value of each product by the ℓ_1 norm of $\mathbf{z}$, which gives L1p its name. We make the exponent learnable because the sharpness required by a layer for a specific task is not known before training, while its initial value serves as a hyperparameter that sets the starting sharpness. Each layer has a single exponent shared across all of its heads, adding only one parameter per layer. The heads can nevertheless differ in sharpness through the relative magnitudes of their query–key products, which evolve during training.

For any two positions i and j with non-zero products, Eq. 1 gives $a_j/a_i = (|z_j|/|z_i|)^p$. This ratio reveals two properties. First, the exponent directly controls sharpness. For $p > 0$ and $|z_j| > |z_i|$, the ratio grows with p, so a larger exponent yields a sharper row and a smaller exponent a flatter one. Second, the ratio depends on the query–key products only through their relative magnitudes. Multiplying every product in $\mathbf{z}$ by any constant $c > 0$ multiplies both the numerator and denominator of Eq. 1 by c^p, leaving every attention score unchanged. We call this property *magnitude invariance*: $\mathbf{a}(c\mathbf{z}) = \mathbf{a}(\mathbf{z})$ for every $c > 0$ and $\mathbf{z} \neq \mathbf{0}$, and say that L1p is *magnitude-invariant*.

Mathematically, L1p is positively homogeneous of degree zero. Unlike the sharpness of softmax, with or without a learnable temperature, that of L1p cannot be changed through the overall magnitude of the query–key products. Consequently, changes in this magnitude cannot offset the effect of the exponent on the sharpness of L1p.

Although the unsigned form of L1p addresses both limitations of softmax, a third limitation remains common to both. Consider the tokens in a row that do not carry information needed for the prediction, which we call *irrelevant*, and those that do, which we call *relevant*. Like softmax, unsigned L1p assigns non-negative scores that sum to 1, making the head output a weighted average of the value vectors. We refer to the share of that total assigned to a set of tokens as its *attention mass*. Any attention mass not assigned to relevant tokens is distributed among irrelevant ones. In a long sequence, many individually small scores on irrelevant tokens can collectively account for a substantial share of this mass. Their weighted value vectors then enter the head output consumed by subsequent layers, potentially mixing irrelevant information into the representation used for prediction. Reducing this contribution under unsigned L1p, as under softmax, generally requires concentrating more attention on relevant tokens.

Such concentration, however, comes at a cost. As a row becomes sharper, irrelevant tokens receive increasingly small scores relative to the others. For L1p with $p > 1$, the derivative of every attention score with respect to a query–key product z_j contains the factor $|z_j|^{p-1}$. This factor is smallest for tokens whose query–key products have small magnitude and vanishes as z_j approaches zero. The gradient reaching such products is correspondingly small, making it difficult for a token that becomes relevant later in training to acquire a large score. Sparsemax (Martins & Astudillo, 2016) and α-entmax (Peters et al., 2019) with $\alpha > 1$, both designed to induce sparse attention, represent an extreme form of this behavior. In particular, they can assign a token a score of exactly zero, in which case its query–key product receives zero local gradient.

To reduce the contribution of irrelevant tokens without necessarily shrinking their score magnitudes, we define a second form of L1p, which we term the *signed* form. It multiplies each unsigned attention score by the sign of the respective query–key product:

$$a_j = \frac{\operatorname{sign}(z_j)\,|z_j|^p}{\sum_{l=1}^{n} |z_l|^p}. \tag{2}$$

A score is negative whenever its query–key product is negative, causing the corresponding value vector to be subtracted from the head output rather than added. Although the magnitudes of the scores still sum to 1, the scores themselves can sum to any value in $[-1, 1]$. The head output is consequently no longer necessarily a weighted average of the value vectors and can lie outside their convex hull. Moreover, contributions from two tokens with opposite-signed scores partially cancel when their value vectors have a positive inner product. The pair can thus contribute less to the head output than under the unsigned form without reducing either score magnitude. Magnitude invariance and control of sharpness through the exponent p carry over unchanged to the signed form.

In our experiments, we observe that using the signed form in more layers increases reasoning depth but also increases language modeling perplexity (see Appendix E.5). We hypothesize that the gain in reasoning depth arises from cancellation, which can suppress the aggregate contribution of irrelevant tokens without forcing their score magnitudes toward zero. We further hypothesize that, in language modeling, information relevant to a prediction is more often distributed across multiple tokens than in our multi-hop reasoning task. Accordingly, cancellation may remove some information needed for prediction, increasing perplexity.

Neither form performs best on both tasks, but a model can combine them across layers. The form used by each layer is an architectural choice fixed before training, and we select the number of signed layers through a sweep (see Appendix E.5). The resulting models use the unsigned form in the first half of their layers and the signed form in the second half. Placing the signed form in the second half rather than the first yields substantially greater reasoning depth (see Appendix E.4). With this arrangement, L1p achieves greater reasoning depth than every normalizing function we compare against (see Section 5.3) while matching the language modeling perplexity of softmax to within a pre-specified margin (see Section 5.4).

4.2 Implementation of L1p

Directly raising query–key products to a large exponent can cause floating-point overflow or underflow, and when all products in a row are small, it can yield the undefined ratio $0/0$. We instead compute L1p through the logarithms of the product magnitudes (Gelada et al., 2025). For the unsigned form, we add a small positive constant ε to each magnitude and compute

$$a_j \;=\; \frac{(|z_j|+\varepsilon)^p}{\sum_{l=1}^{n}(|z_l|+\varepsilon)^p} \;=\; \frac{\exp\big(p\log(|z_j|+\varepsilon)\big)}{\sum_{l=1}^{n}\exp\big(p\log(|z_l|+\varepsilon)\big)}. \tag{3}$$

The equality follows from $\exp(p\log x) = x^p$ for $x > 0$. The right-hand side has the form of softmax applied to $p\log(|z_l|+\varepsilon)$ rather than directly to the query–key products. Standard softmax implementations subtract the largest input before exponentiation, preventing overflow and ensuring a non-zero denominator. The constant ε keeps the logarithm finite when a product is zero. The signed form multiplies each score from Eq. 3 by $\mathrm{sign}(z_j)$.

Nonetheless, training this implementation raises two issues. The first arises when the query–key products become small relative to ε. When they are large relative to ε, the constant has negligible effect. As they become small, however, every term $(|z_l|+\varepsilon)^p$ approaches ε^p, causing the attention scores to become nearly equal regardless of the exponent and thereby flattening the row.

Nothing in the ideal magnitude-invariant form of L1p prevents the query–key products from shrinking. Softmax requires sufficiently separated query–key products to produce sharp attention and can therefore benefit from increasing their scale during training. L1p has no analogous dependence on overall magnitude. Weight decay, by contrast, pulls the query and key projections toward zero and can shrink the products until ε materially affects the computation.

To prevent this, we normalize each row before applying Eq. 3, as Zimerman et al. (2026) did, but by a different row statistic. In particular, we divide the products by their root mean square $s = (n^{-1}\sum_{l=1}^{n} z_l^2)^{1/2}$, while treating s as a constant during backpropagation. Ignoring ε, this replaces $\mathbf{z}$ by $c\mathbf{z}$ with $c = 1/s$, which changes neither the scores of magnitude-invariant L1p nor their gradient with respect to the original query–key products. We treat s as a constant because the computation with ε is not exactly magnitude-invariant, so the division would otherwise contribute a gradient of its own. With ε present, the normalization keeps the computation close to the magnitude-invariant ideal. Every rescaled row has root mean square 1, so at least one product has magnitude at least 1, precluding the entire row from collapsing into the ε-dominated regime.

The second issue concerns the exponent. Like other learnable parameters, it would be pulled toward zero by weight decay. While weight decay regularizes the model's learned projections, applying it to p directly regularizes sharpness. As p approaches zero, the score magnitudes become increasingly similar, and at $p = 0$ they are all equal. Weight decay would thus pull L1p toward flatness regardless of the sharpness the task requires. We consequently exclude the exponents from weight decay.

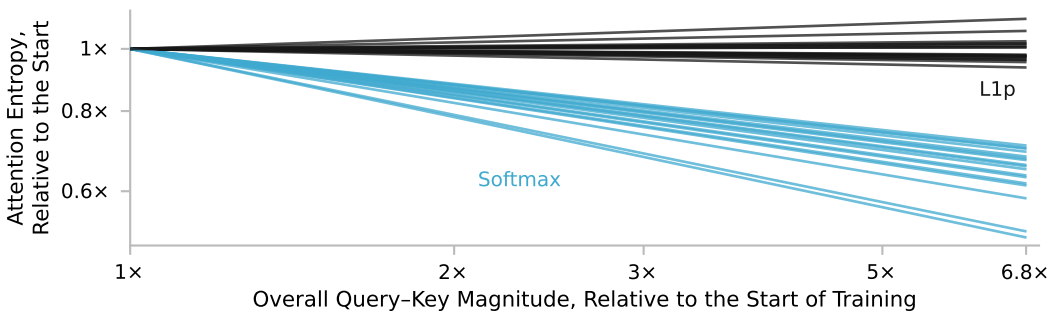

Figure 2: Attention entropy versus overall query–key magnitude during language modeling training. Lower entropy indicates sharper attention; both axes are logarithmic. Each line shows one run's fitted slope from the hyperparameter grid (18 L1p, 17 non-divergent softmax), translated to a common origin over a 6.8-fold magnitude increase that every run reaches at its peak.

5 Experiments

5.1 Experimental Setup

We compare L1p with softmax and five established normalizing functions summarized in Table 1 on a synthetic multi-hop reasoning task and language modeling. We also construct a control, *signed softmax*, which applies softmax to the magnitudes of the query–key products and restores their signs. This tests whether negative attention scores alone suffice to reproduce L1p's reasoning depth, or whether its advantage also depends on properties beyond signed attention.

All models are decoder-only Transformers. The main comparisons use 12-layer models with 55M parameters on multi-hop reasoning and 152M on language modeling; within each task, models differ only in the normalizing function and its tuned hyperparameters. On reasoning, we additionally compare L1p and softmax at 4, 8, 16, and 20 layers. On language modeling, we also compare them at five further model sizes from 76M to 387M parameters. Across all layer counts, L1p uses its unsigned form in the first half of the layers and its signed form in the second half, a design selected through the sweep in Appendix E.5. Appendix A gives the full model and hyperparameter specifications, while Appendix F compares the training and inference speed of L1p and softmax.

The reasoning task is adapted from Allen-Zhu (2025). Each sequence lists directed-graph edges followed by questions specifying only a starting node and number of hops, so the traversal must be composed within a single forward pass. We rescale accuracy from guessing (0) to perfect (100) and deem a hop count solved at 50. We use 4 hops as the primary comparison and further report results at 1, 2, 8, and 16 hops. Appendix C gives full task and scoring details.

For language modeling, models are trained on the 10-billion-token FineWeb sample (Penedo et al., 2024) and evaluated by held-out perplexity (Jelinek et al., 1977). Our goal is to determine whether L1p preserves softmax's language modeling performance. We say that two normalizing functions *match* when the 95% confidence interval on their perplexity difference lies within a pre-specified ± 0.20 margin, equivalent to the two one-sided tests procedure (Schuirmann, 1987) at 2.5% per test. We also evaluate 20 downstream tasks and use perplexity as the primary metric in this section. Appendix D.4, however, reports full results for L1p, softmax, α-entmax, and Scaled Signed Averaging (under both our tuned and the authors' settings), including accuracies on all 20 tasks.

Our main comparisons use 32 random seeds per normalizing function on the reasoning task and up to 24 on language modeling. All presented p-values are two-sided. We use Fisher's exact test to compare the number of runs solving 4 hops, and the Cochran–Armitage test to assess trends in this count with the number of layers or signed layers. We compare perplexities, accuracies, and measures of attention sharpness through differences in means with 95% confidence intervals and Welch's t-tests. Appendix A gives the seed counts for every experiment and the full statistical procedures.

5.2 Input Magnitude and Sharpness

Sections 3 and 4 show that scaling the query–key products sharpens softmax attention while leaving magnitude-invariant L1p unchanged. We measure sharpness by *attention entropy*, the normalized entropy of the attention-score magnitudes, with lower entropy indicating sharper attention. Our analysis covers 32 runs each of L1p and softmax on the reasoning task, and 18 L1p and 17 non-divergent

softmax runs on language modeling, drawn from the hyperparameter grid (see Appendix A). Because both attention entropy and query–key magnitude evolve during training, for each run we fit the slope of log attention entropy against log overall magnitude. A negative slope means that attention becomes sharper as magnitude grows, while zero indicates no association.

On both tasks, *every* slope of L1p lies above *every* slope of softmax. Fig. 2 shows the 35 language modeling runs; Fig. 4 in Appendix B reports the fitted slope of every run on both tasks. On language modeling, the mean slopes are -0.003 for L1p and -0.23 for softmax, with a 95% confidence interval of $[+0.20, +0.26]$ on the difference and $p = 3.2 \times 10^{-14}$. On the reasoning task, the corresponding means are $+0.10$ and -0.17, with an interval of $[+0.24, +0.29]$ and $p = 3.1 \times 10^{-30}$.

Because training changes all model parameters simultaneously, these within-run slopes measure association rather than the causal effect of query–key magnitude on sharpness. In particular, L1p is magnitude-invariant, so its non-zero slopes must arise from other changes during training. The comparison thus establishes that the association between magnitude and sharpness differs between L1p and softmax, consistent across both tasks. Appendix B defines the measurement in full, reports every slope, and shows that sharpness alone does not account for reasoning depth.

Table 2: Reasoning depth and language modeling results. Reasoning entries give the number of runs, out of 32, solving each hop count; perplexity is the mean ± standard deviation over 24 seeds (16 where marked †). Boldface marks the highest non-zero count in each hop column and the lowest perplexity.

Normalizing function	1 hop	2 hops	4 hops	8 hops	16 hops	**Perplexity**
L1p	**32**	**32**	**18**	**4**	0	23.20 ± 0.16
Softmax	5	2	0	0	0	23.23 ± 0.17
Cog Attention	**32**	24	3	0	0	$23.31 \pm 0.53^{\dagger}$
ℓ_1 normalization	0	0	0	0	0	$27.91 \pm 0.11^{\dagger}$
Sparsemax	0	0	0	0	0	$23.01 \pm 0.22^{\dagger}$
α-entmax	5	2	0	0	0	$\mathbf{22.78 \pm 0.20}$
Scaled Signed Averaging	1	0	0	0	0	24.55 ± 0.22
Signed softmax	29	20	1	0	0	$23.57 \pm 0.40^{\dagger}$

5.3 Reasoning Depth

We measure reasoning depth by the number of runs that solve each hop count. As shown in Table 2, at 4 hops, L1p solves the task in 18 of 32 runs, compared with 3 for Cog Attention ($p = 1.2 \times 10^{-4}$), 1 for signed softmax ($p = 3.5 \times 10^{-6}$), and none for the other five functions ($p = 2.6 \times 10^{-7}$ each). At 8 hops, only L1p solves the task, in 4 runs ($p = 0.11$ each); no function solves 16 hops.

Signed softmax, Cog Attention, and ℓ_1 normalization all permit negative attention scores, yet none solves 4 hops in more than 3 of 32 runs. Negative attention scores alone are therefore insufficient to reproduce L1p's reasoning depth. Appendix E provides additional ablations of the signed form.

Sanford et al. (2024) showed that logarithmically many layers suffice for their k-hop task. Consequently, we compare L1p and softmax at 4, 8, 12, 16, and 20 layers, with 16 seeds each. The number of L1p runs solving 4 hops rises from $0/16$ at 4 layers to $15/16$ at 20, with $z = 5.54$ and $p = 3.1 \times 10^{-8}$ for the trend. Softmax solves 4 hops in no run at any tested layer count.

All counts are measured under a fixed training budget. The best 20-layer softmax run reaches a rescaled 4-hop accuracy of 46, just below the solving threshold of 50, and is still improving when training ends. The results thereby measure reasoning depth reached within this budget. Appendix C reports all hop and layer counts and the sensitivity of the 4-hop results to the solving threshold.

5.4 Language Modeling

Table 2 reports held-out perplexity at 152M parameters and shows that L1p matches softmax (see also Fig. 3 (a)). Across 24 seeds, L1p reaches 23.20 ± 0.16 and softmax 23.23 ± 0.17, with a difference of -0.03, a 95% confidence interval of $[-0.13, +0.07]$, and $p = 0.55$. The interval lies entirely within the pre-specified ± 0.20 margin and contains zero, establishing a match without showing that either function has lower perplexity. We also evaluate L1p, softmax, α-entmax, and Scaled Signed Averaging at both of its settings on 20 downstream benchmark tasks (see Appendix D.4). On 12 of these tasks, the best-scoring normalizing function can be distinguished from chance in a single run.

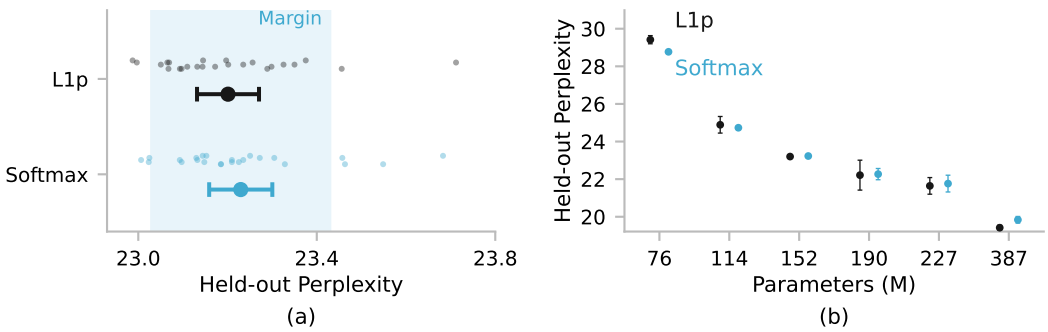

Figure 3: Held-out perplexity of L1p and softmax. (a) All 24 runs per function at 152M; points are runs, solid markers are means with 95% confidence intervals, and shading spans ± 0.20 around the softmax mean. (b) Mean perplexity across six model sizes with 95% confidence intervals; seeds are 24 at 152M, 8 at 387M, and 3 otherwise. The 152M match is based on the confidence interval on the difference, $[-0.13, +0.07]$.

Table 3: Design choices of L1p and the appendix experiments supporting them. Some experiments use settings different from those of the final model, as noted in the corresponding appendices.

Question	**Answer**	**Appendix**
Should the attention scores be computed using a power or an exponential of the magnitudes of the query–key products?	Power	E.1
Should the exponent p be learned during training or held fixed?	Learned	E.2
At what value should the exponent p be initialized?	6	E.3
Should any layer use the signed form of L1p (Eq. 2)?	Yes	E.4, E.5
How many of the 12 layers should use the signed form?	6	E.5
Should the 6 layers using the signed form be the first 6 or the last 6?	The last 6	E.4

Comparing L1p and softmax on these 12 tasks, the interval on their difference excludes zero on 4, split evenly, but only the 2 favoring softmax survive a Holm correction.

No normalizing function with lower perplexity than L1p reaches its reasoning depth. α-entmax has the lowest perplexity, 0.45 below softmax, with a confidence interval of $[-0.56, -0.34]$ and $p = 1.1 \times 10^{-10}$, yet solves 4 hops in none of its 32 runs, compared with 18 for L1p. Table 16 in Appendix D reports the full perplexity comparison, including each normalizing function's difference from softmax, its 95% confidence interval, and its p-value.

We further compare L1p and softmax at five additional model sizes (see Fig. 3 (b)). At 76M, L1p has 0.64 higher perplexity, with an interval of $[+0.46, +0.83]$ and $p = 2.0 \times 10^{-3}$; at 387M, it has 0.42 lower perplexity, with an interval of $[-0.62, -0.22]$ and $p = 5.7 \times 10^{-4}$. At 114M, 190M, and 227M, the intervals contain zero but extend beyond ± 0.20, establishing neither a match nor a difference. Appendix D reports the complete results at all six model sizes, including confidence intervals and p-values.

5.5 Design Choices and Ablations

Table 3 summarizes the design choices of L1p, with the supporting ablations and tradeoffs reported in the corresponding appendices.

6 Conclusion

In this paper, we introduced powered L1 attention (L1p), a normalizing function whose sharpness is controlled by a dedicated parameter rather than determined by the overall magnitude of its inputs. L1p combines three properties: (1) sharpness controlled by a learnable per-layer exponent; (2) attention scores invariant to the overall magnitude of the query–key products; and (3) a signed form that permits negative attention scores. In our main comparisons, L1p reaches greater reasoning depth than softmax and five established alternative normalizing functions on a synthetic multi-hop task within the same training budget, while matching the language modeling perplexity of softmax.

AI USE STATEMENT

In this work, we used generative AI tools to implement the code that trains and evaluates the models and the scripts that compute our results and draw our figures, to provide feedback on our experimental methodology, to help interpret results, to identify relevant literature and format references, and to assist with editing this paper.

We have not used generative AI tools to propose or refine hypotheses, to formulate mathematical claims, to develop theoretical models or conceptual frameworks, to generate synthetic datasets, or to clean or reformat datasets. The writing of proofs, translation, and qualitative or thematic data analysis are not applicable to this work.

We have reviewed all AI-assisted work. Every quantitative result in this paper is recomputed from the logged experiments by version-controlled scripts, which we reviewed and tested. The research question, the hypotheses and evaluation criteria, the comparisons, and the design of L1p are our own. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 4 and Appendix A.1 specify L1p in full. Appendices A–F describe the experimental protocol and report every comparison in this paper.

The source code is attached as supplementary material. It contains the implementation of L1p and the normalizing functions we compare against, the synthetic multi-hop reasoning task, the language modeling task, and the training and evaluation scripts for every model in this paper. It also contains the configuration files for all reported experiments, the recorded measurements of these runs, the scripts that compute the reported statistics, and the scripts that draw the figures and tables from these measurements. The reported results can therefore be regenerated without training a model. Reproducing the measurements themselves requires training the corresponding models. Because training is not deterministic, a re-run is expected to reproduce the means over seeds and the numbers of runs that solve the reasoning task within their reported uncertainty, rather than the values of individual runs.

REFERENCES

Zeyuan Allen-Zhu. Physics of Language Models: Part 4.1, architecture design and the magic of Canon layers. In *Advances in Neural Information Processing Systems*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/3c4141c12660ad3625eb4ae845e0a6f9-Abstract-Conference.html. Full version: arXiv:2512.17351.

Ben Anson and Laurence Aitchison. Controlling changes to attention logits, 2025. URL https://arxiv.org/abs/2511.21377.

Ben Anson, Xi Wang, and Laurence Aitchison. Scale-invariant attention. In *Advances in Neural Information Processing Systems*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/15ddb1773510075ef44981cdb204330b-Abstract-Conference.html.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Nghia Bui, Guergana Savova, and Lijing Wang. Assessing the macro and micro effects of random seeds on fine-tuning large language models. In *International Joint Conference on Natural Language Processing and Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pp. 41–46, 2025. URL https://arxiv.org/abs/2503.07329.

Dallas Card, Peter Henderson, Urvashi Khandelwal, Robin Jia, Kyle Mahowald, and Dan Jurafsky. With little power comes great responsibility. In *Conference on Empirical Methods in Natural Language Processing*, 2020. URL https://arxiv.org/abs/2010.06595.

C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934. doi: 10.1093/biomet/26.4.404. URL https://doi.org/10.1093/biomet/26.4.404.

Gonçalo M. Correia, Vlad Niculae, and André F. T. Martins. Adaptively sparse Transformers. In *Conference on Empirical Methods in Natural Language Processing*, 2019. URL https://arxiv.org/abs/1909.00015.

Alicia Curth, Rachel Lawrence, Sushrut Karmalkar, and Niranjani Prasad. Do Transformers use their depth adaptively? Evidence from a relational reasoning task. In *ICLR Workshop on Logical Reasoning of Large Language Models*, 2026. URL https://arxiv.org/abs/2604.12426.

Mostafa Dehghani, Josip Djolonga, Basil Mustafa, Piotr Padlewski, Jonathan Heek, Justin Gilmer, Andreas Steiner, Mathilde Caron, Robert Geirhos, Ibrahim Alabdulmohsin, Rodolphe Jenatton, Lucas Beyer, Michael Tschannen, Anurag Arnab, Xiao Wang, Carlos Riquelme, Matthias Minderer, Joan Puigcerver, Utku Evci, Manoj Kumar, Sjoerd van Steenkiste, Gamaleldin F. Elsayed, Aravindh Mahendran, Fisher Yu, Avital Oliver, Fantine Huot, Jasmijn Bastings, Mark Patrick Collier, Alexey Gritsenko, Vighnesh Birodkar, Cristina Vasconcelos, Yi Tay, Thomas Mensink, Alexander Kolesnikov, Filip Pavetić, Dustin Tran, Thomas Kipf, Mario Lučić, Xiaohua Zhai, Daniel Keysers, Jeremiah Harmsen, and Neil Houlsby. Scaling vision Transformers to 22 billion parameters. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2302.05442.

Yichuan Deng, Zhao Song, Kaijun Yuan, and Tianyi Zhou. Why softmax attention outperforms linear attention. In *IEEE International Conference on Big Data*, pp. 16–25, 2025. doi: 10.1109/BigData66926.2025.11400978. URL https://arxiv.org/abs/2310.11685.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. Show your work: Improved reporting of experimental results. In *Conference on Empirical Methods in Natural Language Processing*, 2019. URL https://arxiv.org/abs/1909.03004.

Philipp Dufter, Martin Schmitt, and Hinrich Schütze. Increasing learning efficiency of self-attention networks through direct position interactions, learnable temperature, and convoluted attention. In *International Conference on Computational Linguistics*, pp. 3630–3636, 2020.

Sai Surya Duvvuri, Chanakya Ekbote, Rachit Bansal, Rishabh Tiwari, Devvrit Khatri, David Brandfonbrener, Paul Liang, Inderjit Dhillon, and Manzil Zaheer. Interleaved head attention, 2026. URL https://arxiv.org/abs/2602.21371.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The Language Model Evaluation Harness, May 2026. URL https://doi.org/10.5281/zenodo.20122284.

Carles Gelada, Jacob Buckman, Sean Zhang, and Txus Bach. Scaling context requires rethinking attention, 2025. URL https://arxiv.org/abs/2507.04239.

Nuno Gonçalves, Marcos Treviso, and André F. T. Martins. AdaSplash: Adaptive sparse flash attention. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2502.12082.

Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for Transformers. In *Findings of the Association for Computational Linguistics: EMNLP*, 2020. URL https://arxiv.org/abs/2010.04245.

Andreas Hochlehnert, Hardik Bhatnagar, Vishaal Udandarao, Samuel Albanie, Ameya Prabhu, and Matthias Bethge. A sober look at progress in language model reasoning: Pitfalls and paths to reproducibility. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2504.07086.

David T. Hoffmann, Simon Schrodi, Jelena Bratulić, Nadine Behrmann, Volker Fischer, and Thomas Brox. Eureka-moments in Transformers: Multi-step tasks reveal softmax induced optimization problems. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2310.12956.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2404.06654.

F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63, 1977. doi: 10.1121/1.2016299. URL https://doi.org/10.1121/1.2016299.

Zixuan Jiang, Jiaqi Gu, and David Z. Pan. NormSoftmax: Normalizing the input of softmax to accelerate and stabilize training. In *IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–6. IEEE, 2023.

Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. PolySketchFormer: Fast Transformers via sketching polynomial kernels. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2310.01655.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, et al. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.01117.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Jiecheng Lu, Xu Han, Yan Sun, Viresh Pati, Yubin Kim, Siddhartha Somani, and Shihao Yang. ZeroS: Zero-sum linear attention for efficient Transformers. In *Advances in Neural Information Processing Systems*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/1363163299a172662dcf0c0f9932acf6-Abstract-Conference.html.

Ang Lv, Ruobing Xie, Shuaipeng Li, Jiayi Liao, Xingwu Sun, Zhanhui Kang, Di Wang, and Rui Yan. More expressive attention with negative weights, 2024. URL https://arxiv.org/abs/2411.07176.

André F. T. Martins and Ramón Fernandez Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, 2016. URL https://arxiv.org/abs/1602.02068.

Weikang Meng, Yadan Luo, Xin Li, Dongmei Jiang, and Zheng Zhang. PolaFormer: Polarity-aware linear attention for vision Transformers. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kN6MFmKUSK.

Timur Mudarisov, Mikhail Burtsev, Tatiana Petrova, and Radu State. Limitations of normalization in attention. In *Advances in Neural Information Processing Systems*, 2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/8d6fa78de53d5f24b67faf51aec86bbc-Abstract-Conference.html.

Omar Naim, Swarnadeep Bhar, Jérôme Bolte, and Nicholas Asher. SSA: Improving performance with a better scoring function. In *Annual Meeting of the Association for Computational Linguistics*, pp. 30033–30051, 2026. URL https://arxiv.org/abs/2508.14685.

Ken M. Nakanishi. Scalable-Softmax is superior for attention, 2025. URL https://arxiv.org/abs/2501.19399.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.17557.

Binghui Peng, Srini Narayanan, and Christos Papadimitriou. On limitations of the Transformer architecture. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2402.08164.

Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In *Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://arxiv.org/abs/1905.05702.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. In *ICLR Workshop on the Role of Mathematical Reasoning in General Artificial Intelligence*, 2021. URL https://arxiv.org/abs/2201.02177.

Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear Transformer. In *Conference on Empirical Methods in Natural Language Processing*, pp. 7025–7041, 2022. URL https://arxiv.org/abs/2210.10340.

Oliver Richter and Roger Wattenhofer. Normalized attention without probability cage, 2020. URL https://arxiv.org/abs/2005.09561.

Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2402.09268.

Hemanth Saratchandran, Jianqiao Zheng, Yiping Ji, Wenbo Zhang, and Simon Lucey. Rethinking attention: Polynomial alternatives to softmax in Transformers, 2024. URL https://arxiv.org/abs/2410.18613.

Donald J. Schuirmann. A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability. *Journal of Pharmacokinetics and Biopharmaceutics*, 15(6):657–680, 1987.

Noam Shazeer. GLU variants improve Transformer, 2020. URL https://arxiv.org/abs/2002.05202.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced Transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. URL https://arxiv.org/abs/2104.09864.

Hui Tian, Zheng Qin, Renjiao Yi, Chenyang Zhu, and Kai Xu. Tensorformer: Normalized matrix attention Transformer for high-quality point cloud reconstruction. *IEEE Transactions on Multimedia*, 27:718–730, 2025. doi: 10.1109/TMM.2023.3277271. URL https://arxiv.org/abs/2306.15989.

Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency, 2023. URL https://arxiv.org/abs/2309.02390.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Pavlo Vasylenko, Hugo Pitorro, André F. T. Martins, and Marcos Treviso. Long-context generalization with sparse attention. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2506.16640.

Petar Veličković, Christos Perivolaropoulos, Federico Barbero, and Razvan Pascanu. Softmax is not enough (for sharp size generalisation). In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2410.01104.

Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6:287–302, 2018.

Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie Everett, Alex Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale Transformer training instabilities. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2309.14322.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

Tianzhu Ye, Li Dong, Yuqing Xia, Yutao Sun, Yi Zhu, Gao Huang, and Furu Wei. Differential Transformer. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.05258.

Shuangfei Zhai, Tatiana Likhomanenko, Etai Littwin, Dan Busbridge, Jason Ramapuram, Yizhe Zhang, Jiatao Gu, and Josh Susskind. Stabilizing Transformer training by preventing attention entropy collapse. In *International Conference on Machine Learning*, 2023. URL https://proceedings.mlr.press/v202/zhai23a.html.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1910.07467.

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2402.04347.

Itamar Zimerman, Allon Adir, Ehud Aharoni, Matan Avitan, Moran Baruch, Nir Drucker, Jenny Lerner, Ramy Masalha, Reut Moshe, and Omri Soceanu. PowerSoftmax: Towards secure LLM inference over encrypted data. In *International Conference on Artificial Intelligence and Statistics*, pp. 5203–5211, 2026. URL https://proceedings.mlr.press/v300/zimerman26a.html.

A Experimental Details

This appendix describes the experimental setup for both tasks, including the models, training hyperparameters, hardware and software, statistical procedures, and experimental runs and seeds.

A.1 Models

All models are decoder-only Transformers. Table 4 specifies the architecture used for each task. Within a task, models differ only in their normalizing function. Some normalizing functions introduce additional learnable parameters, but these add at most 192 parameters to a 55M-parameter model. We therefore consider the models within each task parameter-matched.

Table 4: Model architectures used in the main comparison for each task. Entries spanning both columns apply to both tasks. Attention and MLP output projections are initialized with standard deviation $0.02/\sqrt{2L}$, where L is the number of layers, rather than 0.02.

	Multi-hop Reasoning	**Language Modeling**
Layers	12	12
Attention heads	8	12
Embedding dimension	512	768
MLP hidden dimension	2,048	3,072
Vocabulary size	10,000	50,304
Sequence length	1,024	2,048
Parameters	55M	152M
Parameters outside the embedding	50M	113M
MLP	SwiGLU (Shazeer, 2020)	
Position encoding	Rotary (Su et al., 2024), with base 10,000	
Normalization	Pre-norm, with RMSNorm (Zhang & Sennrich, 2019)	
Output head	Tied to the token embedding	
Bias terms	None	
Initialization	Normal, with standard deviation 0.02	

We configure the normalizing functions as follows:

- **L1p** uses the unsigned form (Eq. 1) in its first 6 layers and the signed form (Eq. 2) in its last 6. Each layer has one exponent, shared across all its heads and initialized at 6. We set ε in Eq. 3 to 10^{-6}.
- **Softmax** is used in its standard form in every layer.
- **Cog Attention** (Lv et al., 2024) applies softmax to the magnitudes of the query–key products and multiplies each resulting score by the sign of its product. Its authors retained softmax in the first and last layers, with the number of such layers increasing with model size. For a 12-layer model, they used softmax in one layer at each end. We follow this configuration, using softmax in layers 1 and 12 and Cog Attention in layers 2–11.
- **ℓ_1 normalization** divides each query–key product by the sum of the magnitudes of all products in its row. It applies row-wise the same normalization that Tensorformer (Tian et al., 2025) applies along the channel dimension. Its attention scores keep the signs of the products, and the magnitudes of the scores sum to 1. ℓ_1 normalization is a special case of signed L1p with $p = 1$. We use it in every layer.
- **Sparsemax** (Martins & Astudillo, 2016) is used in every layer as originally defined by its authors.
- **α-entmax** (Peters et al., 2019) is used in every layer with $\alpha = 1.5$. This lies between $\alpha = 1$ and $\alpha = 2$, for which α-entmax reduces to softmax and sparsemax, respectively.
- **Scaled Signed Averaging (SSA)** (Naim et al., 2026) has two parameters, b and n. For their in-context learning tasks, the authors of SSA made both parameters learnable in each

head. For language modeling, they fixed b at 0.8 and made one n learnable per layer. We use the first parameterization on the reasoning task and the second on language modeling. Each learnable SSA parameter is initialized at 1. SSA* denotes SSA trained using the hyperparameters reported by its authors.

- **Signed softmax** is the control introduced in Section 5.1. In every layer, it applies softmax to the magnitudes of the query–key products and multiplies each resulting score by the sign of its product. It thus adds signs to magnitude-based softmax in the same way that signed L1p adds them to unsigned L1p. It computes the same scores as Cog Attention but applies them in every layer, including the first and last.

For the design choices examined in Section 5.5, we also train three alternative mechanisms for controlling sharpness. We define and configure them as follows:

- **The exponential form** divides the magnitudes of the query–key products by their root mean square across the row, multiplies them by a learnable inverse temperature, and applies softmax. Like L1p, it multiplies each score by the sign of its product only in the last 6 layers. Backpropagation passes through the root mean square, so that, as in L1p, the gradient with respect to the products has no component along the row of products. Each layer has one inverse temperature, initialized at 8.
- **Magnitude-invariant softmax** divides the query–key products themselves by their root mean square over the row, multiplies them by a learnable inverse temperature, and applies softmax. As in L1p, the root mean square is treated as a constant during backpropagation. Because softmax is not magnitude-invariant on its own, the gradient with respect to the products then has a component along the row of products, which changes their overall magnitude without affecting the scores. We use it in every layer. Each layer has one inverse temperature, initialized at 4.
- **NormSoftmax** (Jiang et al., 2023) divides the query–key products by their standard deviation over the row, capped at a fixed threshold, and applies softmax. We use it in every layer, with the threshold set to 1.

A.2 Training hyperparameters

Table 5 lists the training hyperparameters for each task. We exclude the learnable parameters of all normalizing functions from weight decay, for the same reason given in Section 4.2 for the exponents of L1p.

Table 5: Training hyperparameters on each task. An entry that spans both columns applies to both tasks.

	Multi-hop Reasoning	**Language Modeling**
Training data	Generated on the fly	One pass over the corpus
Tokens in the loss	Separator and answer tokens	Every token
Global batch	128 sequences	256 sequences
Training steps	100,000	19,073
Weight decay	0.03	Table 6
Optimizer	AdamW (Loshchilov & Hutter, 2019)	
Optimizer parameters	$\beta = (0.9, 0.98)$, $\epsilon = 10^{-6}$	
Learning rate	Table 6	
Warmup	Linear, over 1,000 steps	
Decay	Cosine, to a tenth of the peak learning rate	
Gradient clipping	To a norm of 1	
Precision	bfloat16 autocast	

We tune the learning rate of each normalizing function separately on both tasks, and its weight decay on language modeling. Each normalizing function is therefore evaluated at its own tuned setting.

Unlike the three alternative mechanisms, the variants of L1p in Appendix E are not tuned. On the reasoning task, they use the learning rate of L1p, so that each variant differs from L1p only in the component it changes. The language modeling experiments in Appendix E use a learning rate of 6×10^{-4} instead, with the weight decay of L1p.

On the reasoning task, we train each normalizing function with 2 seeds at each of four learning rates, 3×10^{-4}, 5×10^{-4}, 1×10^{-3}, and 2×10^{-3}. For the exponential form and magnitude-invariant softmax, we instead try two learning rates, 5×10^{-4} and 1×10^{-3}, each with two initial values of the inverse temperature. These initial values are 4 and 8 for the exponential form, and 1 and 4 for magnitude-invariant softmax. We select the setting with the highest mean rescaled accuracy at the largest number of hops that any run of that normalizing function solves, or at 1 hop if no run solves the task. Rescaled accuracy is 0 for guessing and 100 for answering every question correctly, as Section 5.1 defines it. For softmax, α-entmax, ℓ_1 normalization, sparsemax, SSA, and NormSoftmax, no screening run solves the task even at 1 hop. For SSA and sparsemax, rescaled accuracy at 1 hop also stays below 4 at every learning rate. For these six normalizing functions, we select the rate with the highest accuracy at 1 hop, but these runs do not distinguish it from the other rates. On this task, we tune only the learning rate, and the weight decay and the number of training steps are the same for every normalizing function.

On language modeling, we tune the learning rate and the weight decay of L1p, softmax, α-entmax, and SSA with a grid. The grid contains one run at each combination of 6 learning rates and 3 weight decays. The learning rates run from 6×10^{-4} to 1.92×10^{-2}, each twice the previous one, and the weight decays are 0, 0.05, and 0.1. For the other four normalizing functions (Cog Attention, ℓ_1 normalization, sparsemax, and signed softmax), we tune only the learning rate, with one seed at each of the middle 4 of these learning rates and a weight decay of 0.05. In both cases, we select the setting with the lowest held-out perplexity. We also train SSA using the hyperparameters that its authors reported. Table 6 gives the selected values. On both tasks, these selections use the evaluation data of the main comparisons, but seeds that no main comparison uses.

Table 6: Selected learning rate and weight decay of each normalizing function, including the three alternative mechanisms, which are not in Table 2. The weight decay on the reasoning task is 0.03 for every normalizing function. A dash means that the normalizing function is not trained on that task.

	Multi-hop Reasoning	**Language Modeling**	
Normalizing Function	**Learning Rate**	**Learning Rate**	**Weight Decay**
L1p	5×10^{-4}	4.8×10^{-3}	0.1
Softmax	5×10^{-4}	9.6×10^{-3}	0.05
Cog Attention	5×10^{-4}	4.8×10^{-3}	0.05
ℓ_1 normalization	1×10^{-3}	2.4×10^{-3}	0.05
Sparsemax	3×10^{-4}	2.4×10^{-3}	0.05
α-entmax	5×10^{-4}	4.8×10^{-3}	0.05
SSA	3×10^{-4}	1.2×10^{-3}	0.05
SSA*	—	3×10^{-4}	0.1
Signed softmax	5×10^{-4}	4.8×10^{-3}	0.05
Exponential form	1×10^{-3}	—	—
Magnitude-invariant softmax	1×10^{-3}	—	—
NormSoftmax	1×10^{-3}	—	—

A.3 Hardware and training environment

We train every run on one of four machines with the same configuration. Table 7 gives their hardware and software. On the reasoning task, each run uses one accelerator, and each machine trains 8 runs at a time. On language modeling, each run uses all 8 accelerators of a machine, through PyTorch's DistributedDataParallel. Training all the runs in this paper took approximately $2{,}000$ node-hours, where one node-hour is one hour on one machine.

Table 7: Hardware of each machine and software used for every run. The CUDA version is the one that PyTorch was built with.

Accelerators	8 NVIDIA H100 80GB HBM3
Processor cores	104
System memory	1.08 TB
Operating system	Linux 6.8
Python	3.14.3
PyTorch	2.13.0
CUDA	12.9

A.4 Statistics

Table 8 lists the statistical procedure used for each kind of comparison in this paper. Every p-value we report is two-sided. The only one-sided test, used by the selection rule in Appendix E.5, reports whether a setting passes rather than a p-value. The perplexity of a normalizing function is the mean perplexity across its runs. Two normalizing functions match in perplexity when the 95% confidence interval on their difference lies within ± 0.20. This criterion is equivalent to the two one-sided tests procedure (Schuirmann, 1987), with each test at the 2.5% level. Although seeds are shared within an experiment, no test pairs runs by seed.

Table 8: Statistical procedure for each kind of comparison, and what it reports.

Quantity Compared	**Procedure**	**Reported**
Number of runs that solve the reasoning task, between two normalizing functions or two variants of L1p	Fisher's exact test	p
Trend in the number of runs that solve the reasoning task with the number of layers or of signed layers	Cochran–Armitage test, scoring each setting by its number of layers or of signed layers	z and p from its normal approximation
Fraction of runs that solve the reasoning task	Exact binomial interval (Clopper & Pearson, 1934)	95% interval
Perplexity, benchmark accuracy, and attention sharpness	Difference in means, with Welch's t-test	Difference, 95% confidence interval, and p
Match in perplexity	95% confidence interval on the difference within ± 0.20	Match or no match
Spread of attention entropy across seeds	Brown–Forsythe test	p
Perplexity of a setting in the signed-layer sweep, at most 0.20 above softmax	One-sided t-test at the 0.005 level, with a standard deviation from separate runs	Pass or fail

A.5 Experiments and seeds

Table 9 lists every experiment in this paper, with its number of configurations, seeds per configuration, and total runs. The main comparisons in Sections 5.3 and 5.4 use 32 seeds per normalizing function on the reasoning task and up to 24 on language modeling. We use more seeds on the reasoning task for two reasons. First, solving the task is a binary outcome of each run, so separating normalizing functions by the number of runs that solve it requires more runs than separating them by mean perplexity. Second, a language modeling run requires about twice the compute of a reasoning run. The remaining experiments use fewer seeds to fit the available compute.

On both tasks, a seed determines the model initialization. On the reasoning task, it also determines the training examples. On language modeling, it additionally determines the order in which the model reads the corpus, but not which documents are included.

The experiments with other numbers of layers and the ablations in Table 21 use the first 16 seeds from the main reasoning comparison. The runs used to select the learning rate, weight decay, number of signed layers, and initial exponent use seeds that are not used in any main comparison.

Table 9: Every experiment in this paper, with its number of configurations, seeds per configuration, and runs, excluding runs already counted in a preceding row. All configurations within an experiment use the same seeds, except that the model-size experiment adds 5 seeds at 387M and the initial value of 6 in Appendix E.3 uses an additional 8 seeds. The last column lists the sections reporting each experiment, followed by the appendices.

Experiment	Configurations	Seeds	Runs	Reported in
Multi-hop Reasoning				
Normalizing functions	8	32	256	1, 5.2, 5.3; B, C, E
Number of layers	10	16	128	5.3; C
Ablations	8	16	112	5.5; B, E
Number of signed layers	6	24	144	5.5; E
Initialization of the exponent	6	8	48	5.5; E
Learnable or fixed exponent	2	8	8	5.5; E
Learning rate	4 per function	2	88	A
Total			784	
Language Modeling				
Normalizing functions	5	24	120	5.4; B, D
Four further normalizing functions	4	16	64	5.4; D
Model size	10	3 (8 at 387M)	40	5.4; D
Number of signed layers	6	8	48	5.5; E
Reference for the signed-layer sweep	2	12	24	E
Initialization of the exponent	6	8	40	5.5; E
Learnable or fixed exponent	2	8	8	5.5; E
Learning rate and weight decay	18 per function	1	72	5.2; A, B
Learning rate	4 per function	1	16	A
Learning rate at 387M	3 per function	1	6	D
Half the learning rate at 227M	1 per function	1	2	D
Total			440	
Both tasks			1,224	

B Attention Sharpness

This appendix defines how attention sharpness and the overall magnitude of the query–key products are measured, and reports in full the results summarized in Section 5.2. It then examines how the exponent of L1p changes during training, how much sharpness varies across seeds, and how sharpness relates to reasoning depth.

B.1 Measurement of attention entropy and overall magnitude

We measure attention entropy every 250 training steps on a fixed batch of sequences shared by every run of a task. At each query position, we first renormalize the magnitudes of the attention scores to sum to one, so that signed and unsigned normalizing functions are measured alike. For a row of n attention scores $a_1, \ldots, a_n$, the normalized entropy is

$$H \;=\; -\frac{1}{\log n}\sum_{j=1}^{n} \tilde{a}_j \log \tilde{a}_j, \qquad \tilde{a}_j \;=\; \frac{|a_j|}{\sum_{l=1}^{n} |a_l|}. \tag{4}$$

H is 0 when a single token receives all the attention and 1 when attention is uniform. At the first position, where $n = 1$, we set $H = 0$. Lower attention entropy accordingly indicates sharper attention. The attention entropy of a run at a step is the mean of H over positions, heads, sequences, and layers. Over the corresponding query–key entries and layers, the overall magnitude is the mean of $|z_j|$, measured before L1p divides the products by their root mean square.

B.2 Relation between sharpness and overall magnitude within runs

To relate sharpness to the overall magnitude within a run, we fit the least-squares slope of the logarithm of its attention entropy against the logarithm of its overall magnitude, over the steps of its training. Section 5.2 fits this slope on two sets of runs. On language modeling, the runs are the L1p and softmax runs of the hyperparameter grid in Appendix A.2, one of each at every combination of learning rate and weight decay, all with the same seed. On the reasoning task, the runs are the 32 runs per normalizing function of the main comparison, at a single setting. The first set thus tests whether the difference between L1p and softmax holds across hyperparameter settings, and the second whether it holds across seeds. Fig. 4 shows the slope of every run in both sets.

On language modeling, we compare L1p and softmax within each cell, since the L1p and softmax runs of a cell differ only in the normalizing function. In all 17 cells where neither run diverged, the slope of L1p is closer to zero than that of softmax. The two sets of slopes do not overlap, as the largest slope of softmax is -0.18 and the smallest slope of L1p is -0.03. On the reasoning task, all 32 softmax runs have a negative slope, between -0.40 and -0.12. Only 1 of the 32 L1p runs has a negative slope, at -0.02, and the slopes of L1p range up to $+0.18$. As on language modeling, the two sets do not overlap.

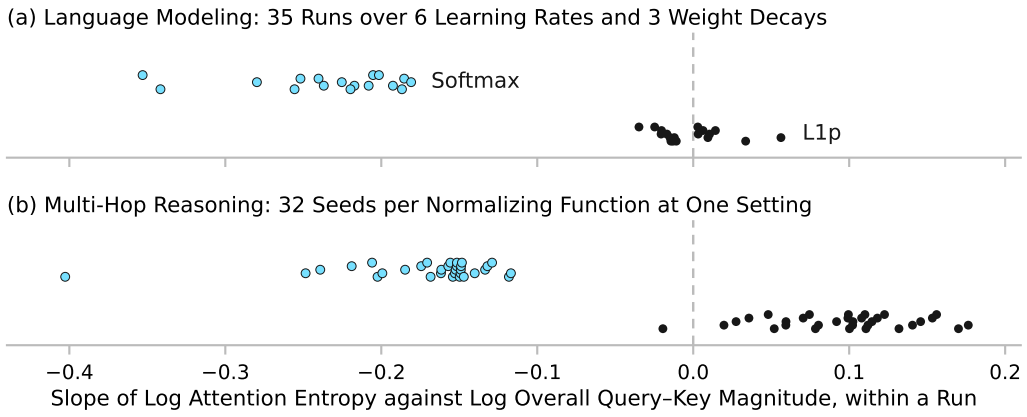

Figure 4: The fitted slope of log attention entropy against log overall magnitude for every run that the measurement in Section 5.2 uses, with each point one run. (a) Language modeling, over 6 learning rates and 3 weight decays at one seed. (b) The multi-hop reasoning task, with 32 seeds per normalizing function at one setting.

B.3 Relation between sharpness and overall magnitude across runs

The hyperparameter grid also shows how sharpness relates to the overall magnitude across runs, because the overall magnitude at the end of training differs widely from one cell to another. Fig. 5 shows the final overall magnitude and the final attention entropy of every L1p and softmax run of the grid. Over the 18 cells, the overall magnitude in the L1p runs ranges from 0.16 to 1,150, a factor of about 7,000, yet the attention entropy of these runs stays between 0.45 and 0.52. In the softmax runs, the overall magnitude ranges only from 2.8 to 4.5 over 17 cells, but the attention entropy ranges from 0.51 to 0.63. The eighteenth softmax run, at the highest learning rate without weight decay, diverged.

How much the overall magnitude of L1p grows during training depends on weight decay, as Table 10 shows. Without weight decay, the overall magnitude of L1p grows by a factor of up to 4,706, whereas with a weight decay of 0.1 it grows by at most 1.9. The attention entropy of these runs nevertheless ends between 0.45 and 0.52 across all weight-decay values. In the softmax runs, by contrast, the overall magnitude grows by a factor of 11.6 to 18.5 at every weight decay, so within this grid weight decay does not offset that growth. Each cell contains a single run of each normalizing function, so we report this comparison without a statistical test.

Table 10: Factor by which the overall magnitude of the query–key products grows during training, from its initial to its final value, over the language modeling hyperparameter grid. Each entry is the range of this factor over the 6 learning rates at one weight decay, with one run per learning rate. A factor below 1 means that the overall magnitude shrinks. Softmax without weight decay has 5 runs, because its run at the highest learning rate diverged.

	Growth of the Overall Magnitude	
Weight Decay	**L1p**	**Softmax**
0	12.1× to 4,706×	11.6× to 14.1×
0.05	4.9× to 9.9×	13.0× to 18.5×
0.1	0.7× to 1.9×	13.7× to 17.7×

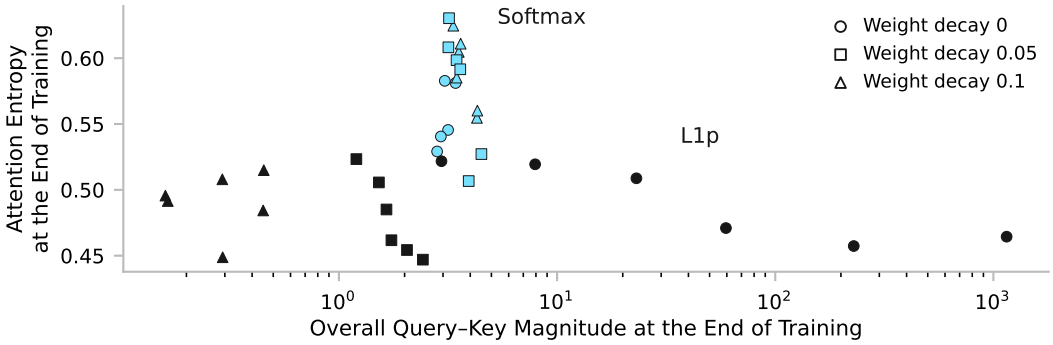

Figure 5: The overall magnitude of the query–key products and the attention entropy at the end of training, for every L1p and softmax run of the language modeling hyperparameter grid, with one seed per cell. Black marks are runs of L1p, blue marks are runs of softmax, and the shape of a mark gives its weight decay. The softmax run at the highest learning rate without weight decay diverged and is not shown.

B.4 Change in the learned exponent during training

The sharpness of L1p is set by its exponent, which each layer learns separately from an initial value of 6. A larger exponent makes attention sharper. Fig. 6 shows the exponent of each layer at the end of training, over the runs of the main comparisons. Averaged over layers, the exponent ends at a median of 8.07 over the 32 reasoning runs and 8.54 over the 24 language modeling runs. This average rises in every run. Training thus moves the exponent toward sharper attention on both tasks. The exponents of different layers, however, change by different amounts. On the reasoning task, the mean exponents of the first three layers end between 5.1 and 6.2, close to their initial value, whereas those of layers 4 to 12 end between 8.2 and 10.3. On language modeling, the mean exponent of every layer ends above its initial value, from 6.9 in the last layer to 9.8 in the sixth.

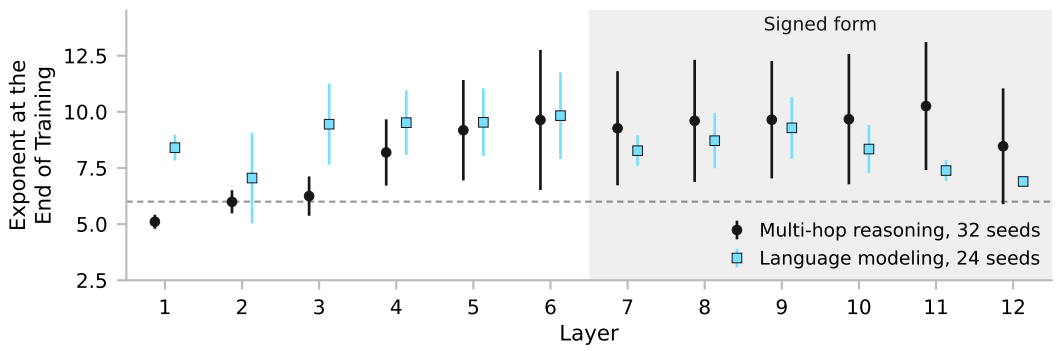

Figure 6: The exponent of L1p in each layer at the end of training, as the mean over the seeds of the main comparisons, with bars of one standard deviation. Every exponent starts at 6, marked by the dashed line. The shaded layers use the signed form.

B.5 Variation of sharpness across seeds

Section 1 illustrates how much softmax runs vary across seeds (see Fig. 1). To determine whether L1p reduces this variation, we compare how much the attention entropy at the end of training varies across the seeds of the main comparisons. Table 11 gives its mean and its spread for L1p and softmax, together with the p-value of the Brown–Forsythe test for each difference in spread. On both tasks, L1p has a lower mean attention entropy than softmax and a smaller standard deviation across seeds.

Two factors limit this comparison. First, the two standard deviations are taken at different means. Relative to its mean, the spread of L1p is also smaller on both tasks, but the same test on the logarithms of the entropies gives $p = 0.39$ and $p = 0.066$. Second, on language modeling the two normalizing functions train at different tuned hyperparameters, unlike on the reasoning task. The difference in their spread therefore cannot be attributed to the normalizing function alone.

Table 11: The attention entropy at the end of training over the runs of the main comparisons, as its mean, its standard deviation across seeds, and that standard deviation relative to the mean. The p-values compare L1p with softmax by the Brown–Forsythe test, applied to the entropies for the standard deviation and to their logarithms for the relative standard deviation.

Task	Normalizing Function	Seeds	Mean	SD	p	SD / Mean	p
Multi-hop reasoning	Softmax	32	0.80	0.063		7.9%	
	L1p	32	0.47	0.027	0.020	5.8%	0.39
Language modeling	Softmax	24	0.54	0.016		2.9%	
	L1p	24	0.46	0.009	0.016	1.9%	0.066

B.6 Relation between sharpness and reasoning depth

Among the softmax runs on the reasoning task, sharpness and rescaled accuracy are associated across seeds, as Fig. 1 shows. The 5 softmax runs that solve the reasoning task at 1 hop end training at a mean attention entropy of 0.69, against 0.82 for the other 27. The difference is -0.13, with a 95% confidence interval of $[-0.20, -0.07]$ and $p = 2.4 \times 10^{-3}$ by Welch's t-test.

Across normalizing functions, however, sharpness alone does not account for reasoning depth. Over the first 16 seeds of the comparison in Section 5.3, Table 12 gives the attention entropy of every normalizing function in Table 2 and of the three alternative mechanisms, together with the number of runs that solve the reasoning task at 4 hops. Sparsemax is the sharpest in Table 12, at an entropy of 0.15 against 0.46 for L1p, yet it solves the task in no run. The exponential form is also slightly

sharper than L1p, at 0.45, and solves the task in only 2 runs, against 9 for L1p. Appendix A.1 describes the three alternative mechanisms.

L1p is sharper than softmax from the start of training. On the reasoning task, at the first probe, taken at step 1, its attention entropy is 0.58, against 1.00 for softmax. At initialization, the division by the root mean square keeps the products of order one, and the exponent, initialized at 6, acts on them directly.

Table 12: The attention entropy at the end of training and the number of runs that solve the multi-hop reasoning task at 4 hops, for every normalizing function in Table 2 and for the three alternative mechanisms. Both use the first 16 seeds of the comparison in Section 5.3. Rows run from the sharpest to the flattest. Boldface marks the largest count.

Normalizing Function	**Attention Entropy**	**Runs that Solve 4 Hops, of 16**
Sparsemax	0.15	0
Exponential form	0.45	2
L1p	0.46	$\mathbf{9}$
Magnitude-invariant softmax	0.48	0
α-entmax	0.56	0
Signed softmax	0.66	1
NormSoftmax	0.68	0
Cog Attention	0.71	2
Softmax	0.82	0
Scaled Signed Averaging	0.89	0
ℓ_1 normalization	0.93	0

C Reasoning Depth

This appendix describes the multi-hop reasoning task in full, including the format of an example, the training and evaluation data, the scoring procedure, and the criterion for solving a hop count. It then gives confidence intervals for the numbers of runs reported in Section 5.3, examines how these numbers depend on the solving threshold, and reports how reasoning depth varies with the number of layers.

C.1 FORMAT OF A REASONING EXAMPLE

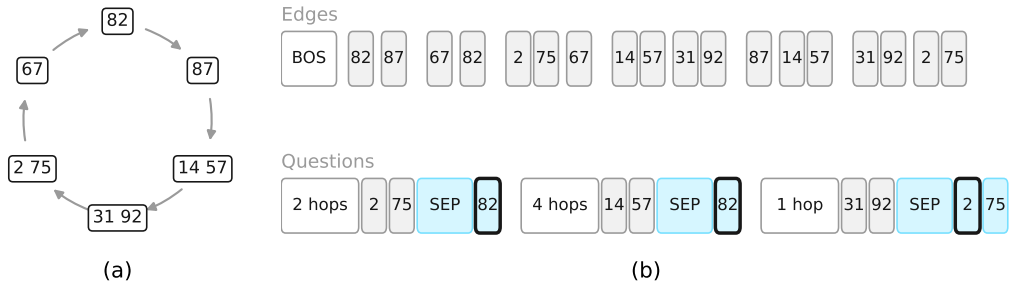

Figure 7: An example of the multi-hop reasoning task generated with a graph of 6 nodes and 3 questions; the evaluation set uses 150 nodes and 10 questions. (a) The directed cycle graph, with each node labeled by its name, a sequence of one or two tokens shown by their indices. (b) The tokens of the example. After a beginning-of-sequence (BOS) token, the edges appear in random order, each as the name of its source node followed by that of its target node. Each question gives a number of hops, a starting node, and a separator (SEP), followed by the answer. Node names are shaded grey, and tokens included in the training loss are shaded blue. The outlined token of each answer is the one scored.

Each example of the reasoning task is generated from a directed cycle graph, in which every node has exactly one outgoing edge. A node is named by 1 or 2 tokens. A two-token name takes its first token from one set of 50 tokens and its second token from a separate set of 50, and a one-token name takes its token from the second set. The last token of every name thereby comes from the second set, allowing the model to identify where each name ends. An example opens with the edges of its graph in random order, each written as the name of its source node followed by the name of its target node. It continues with 10 questions. Each question consists of a token naming a number of hops, the name of a starting node, a separator token, and the name of the node reached from the starting node by following one edge per hop. Fig. 7 shows an example with a smaller graph and fewer questions.

C.2 TRAINING AND EVALUATION DATA

Table 13: The training and the evaluation data of the multi-hop reasoning task. An entry spanning both columns applies to both.

	Training	**Evaluation**
Nodes per graph	3 to 150, favoring small graphs	150
Hops per question	Uniform over 1 to 16	1, 2, 4, 8, or 16
Sequences	Generated on the fly	256, fixed
Questions per example	10	
Tokens per sequence	1,024	

Training and evaluation data differ in graph size and in the numbers of hops requested by their questions, and Table 13 compares the two. During training, the number of nodes N in a graph is drawn with probability proportional to $1/(N+\sqrt{150})$, so that smaller graphs appear more often. The number of hops for each question is drawn uniformly from 1 to 16. The evaluation set is generated once from a single seed and is the same for every run. Its graphs all have 150 nodes, the largest size seen during training. In both cases, a sequence is filled with consecutive examples, and the last example is truncated where the sequence ends. An example with 150 nodes occupies between 555

and 586 tokens and lists its edges before its questions. As a result, every question in a sequence has the entire edge list of its graph in context.

C.3 Scoring and the criterion for solving

An answer is the name of a node, and a name may span two tokens. During evaluation, the model predicts each token of an answer at its position, with the correct tokens up to that point supplied. The second token of a two-token name is consequently predicted with its first token given. That first token already narrows the answer to a median of 3 of the 150 nodes, so predicting the second token requires substantially less of the traversal that the question asks for. We accordingly score the first token alone.

A model that cannot traverse the graph still scores above zero, since it can read the node names from the sequence and name one of them at random. We refer to this as *guessing*. By a Monte Carlo estimate from the task's own generator, guessing is correct on $g = 1.63\%$ of first tokens, and on about 18.2% of all answer tokens. We rescale the first-token accuracy a of a run to

$$r \;=\; 100 \cdot \max\!\left(0,\; \frac{a - g}{1 - g}\right), \tag{5}$$

which is 0 for guessing and 100 for answering every question correctly. A run solves the task at a given number of hops when its rescaled accuracy on questions with that number of hops reaches 50, which corresponds to a first-token accuracy of 50.8%. A run is judged on its accuracy at the end of training rather than on its best value during training, since the best value would be selected on the evaluation set.

C.4 Confidence intervals on the fraction of runs that solve the task

Each count in Table 2 is a number of runs out of 32 and carries the uncertainty of a binomial proportion. Table 14 gives the exact 95% binomial interval (Clopper & Pearson, 1934) on the fraction of runs that solve the task at 4 hops. None of the other intervals overlaps that of L1p.

Table 14: The fraction of runs that solve the multi-hop reasoning task at 4 hops, with its exact 95% binomial interval. The other five normalizing functions (softmax, ℓ_1 normalization, sparsemax, α-entmax, and Scaled Signed Averaging) solve the task in no run.

Normalizing Function	**Runs that Solve 4 Hops, of 32**	**Fraction**	**95% Interval**
L1p	18	0.56	$[0.38, 0.74]$
Cog Attention	3	0.09	$[0.02, 0.25]$
Signed softmax	1	0.03	$[0.00, 0.16]$
Each of the other five	0	0.00	$[0.00, 0.11]$

C.5 Sensitivity of the counts to the threshold for solving

The threshold of 50 lies halfway between guessing and answering every question correctly. That halfway point is a choice, and every count in Table 2 depends on it. Fig. 8 shows how the counts at 4 hops change as the threshold moves from 25 to 75.

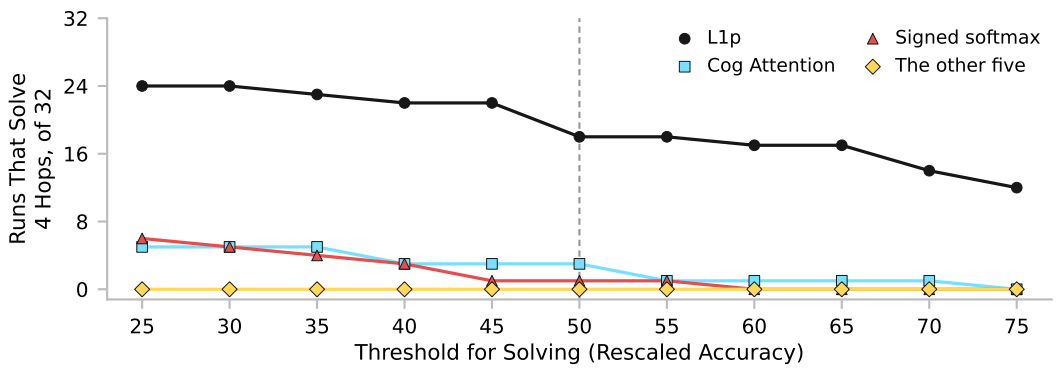

Figure 8: Runs that solve the multi-hop reasoning task at 4 hops, of 32 per normalizing function, as the threshold for solving moves from 25 to 75. The dashed line marks the threshold of 50 used throughout the paper. The other five normalizing functions (softmax, ℓ_1 normalization, sparsemax, α-entmax, and Scaled Signed Averaging) solve the task in no run at any threshold.

Lowering the threshold from 50 to 45, for instance, raises the number of runs of L1p that solve the task from 18 to 22. The four additional runs reach a rescaled accuracy between the two thresholds. At every threshold from 25 to 75, however, L1p solves the task in more runs than any other normalizing function. The largest p-value of Fisher's exact test against any of them, at any threshold, is 2.0×10^{-4}. No other normalizing function solves the task in more than 6 of its 32 runs at any threshold.

C.6 Reasoning depth across numbers of layers

The comparison in Table 2 uses models with 12 layers. Sanford et al. (2024) proved that a number of layers logarithmic in the number of hops suffices for their k-hop task. To examine how reasoning depth depends on the number of layers, we compare L1p with softmax at 4, 8, 12, 16, and 20 layers, with 16 seeds at each. At every number of layers, L1p uses its signed form in the second half of its layers, and both normalizing functions train at a learning rate of 5×10^{-4}. The runs with 12 layers are those of the first 16 seeds of the comparison in Section 5.3. Table 15 reports the runs that solve each number of hops.

Table 15: The number of runs, out of 16, that solve each hop count as a function of the number of model layers. The columns within each normalizing function correspond to hop counts.

	L1p					**Softmax**				
Layers	**1**	**2**	**4**	**8**	**16**	**1**	**2**	**4**	**8**	**16**
4	12	0	0	0	0	1	0	0	0	0
8	16	14	6	0	0	0	0	0	0	0
12	16	16	9	2	0	2	1	0	0	0
16	16	14	11	3	0	6	3	0	0	0
20	16	15	15	6	4	4	2	0	0	0

At 4 hops, the number of runs of L1p that solve the task rises with the number of layers, from 0 of 16 with 4 layers to 15 of 16 with 20 layers. The Cochran–Armitage test gives $z = 5.54$ and $p = 3.1 \times 10^{-8}$ for this trend. No softmax run solves the task at 4 hops at any number of layers. With 20 layers, Fisher's exact test gives $p = 5.7 \times 10^{-8}$ for the difference between 15 of 16 runs of L1p and none of softmax. In accuracy at 4 hops, L1p scores higher in 255 of the 256 comparisons of one run of L1p with one run of softmax.

Adding layers also increases the largest number of hops that L1p solves. With 20 layers, 6 of the 16 runs of L1p solve 8 hops, and 4 solve 16 hops. With fewer layers, no run of L1p solves 16 hops.

The number of softmax runs that solve the task also tends to rise with the number of layers, but only at 1 and 2 hops. At 1 hop, this number is 1, 0, 2, 6, and 4 of 16 over the five numbers of layers, for which the Cochran–Armitage test gives $z = 2.57$ and $p = 0.010$. At 2 hops, the number is 0, 0, 1, 3, and 2, with $z = 2.10$ and $p = 0.036$.

These counts are measured under a fixed training budget. The best softmax run at 20 layers reaches a rescaled accuracy of 46 at 4 hops, below the threshold of 50, and is still improving when training ends. A longer training budget may bring that run above the threshold. Within the range of layers we train, however, the difference between L1p and softmax at 4 hops widens as layers are added.

D Language Modeling

This appendix describes the language modeling data and perplexity measurement, and reports in full the results summarized in Section 5.4. It then evaluates the trained models on downstream benchmarks, examines whether the benchmarks order the normalizing functions as perplexity does, and measures the reliability of a single-run score.

D.1 Training data and held-out perplexity

We train every language model on the 10-billion-token sample of FineWeb (Penedo et al., 2024), tokenized with the GPT-2 tokenizer into sequences of 2,048 tokens. The first 8,000 documents of the corpus are held out from training. At the end of training, we compute the perplexity of each run on 512 sequences of 2,048 tokens drawn from these documents. These sequences are the same for every run. Every perplexity in this appendix is computed in this way, except the perplexities on WikiText and LAMBADA, which the evaluation harness in Appendix D.4 computes.

D.2 Perplexity of every normalizing function at 152M parameters

Table 16 reports the held-out perplexity of every normalizing function at 152M parameters. For each, the table also gives the difference from softmax, with its 95% confidence interval and the p-value of Welch's t-test. L1p matches softmax, since the interval on their difference lies within ± 0.20. α-entmax and Scaled Signed Averaging (SSA) differ from softmax, with intervals that exclude zero. The perplexity of α-entmax is 0.45 lower than that of softmax. That of SSA is 1.32 higher, and 3.68 higher when trained using the hyperparameters its authors reported. The last four normalizing functions in the table are trained with the first 16 of the 24 seeds and tuned only on the learning rate (see Appendix A.2). The perplexity of sparsemax is 0.22 lower than that of softmax, and those of signed softmax and ℓ_1 normalization are 0.34 and 4.68 higher, with intervals that exclude zero. For Cog Attention, the interval on the difference, $[-0.21, +0.37]$, contains zero but extends beyond ± 0.20, establishing neither a match nor a difference.

Table 16: Held-out perplexity of every normalizing function at 152M parameters, as the mean over the seeds given and the standard deviation across them. The difference is the perplexity of each normalizing function minus that of softmax, with its 95% confidence interval and the p-value of Welch's t-test. SSA* is SSA trained using the hyperparameters its authors reported.

Normalizing Function	Seeds	Perplexity	Difference from Softmax	p
Softmax	24	23.23 ± 0.17	—	—
L1p	24	23.20 ± 0.16	-0.03 [-0.13, $+0.07$]	0.55
α-entmax	24	22.78 ± 0.20	-0.45 [-0.56, -0.34]	1.1×10^{-10}
SSA	24	24.55 ± 0.22	$+1.32$ [$+1.21$, $+1.44$]	1.0×10^{-25}
SSA*	24	26.91 ± 0.10	$+3.68$ [$+3.60$, $+3.76$]	2.7×10^{-45}
Cog Attention	16	23.31 ± 0.53	$+0.08$ [-0.21, $+0.37$]	0.56
Signed softmax	16	23.57 ± 0.40	$+0.34$ [$+0.12$, $+0.56$]	4.5×10^{-3}
ℓ_1 normalization	16	27.91 ± 0.11	$+4.68$ [$+4.59$, $+4.77$]	1.2×10^{-48}
Sparsemax	16	23.01 ± 0.22	-0.22 [-0.35, -0.09]	2.2×10^{-3}

D.3 Perplexity of L1p and softmax across model sizes

The comparison in Section 5.4 uses a single model size, 152M parameters. To examine whether the match between L1p and softmax holds at other sizes, we train both at five more sizes. Four of them have 4, 8, 16, and 20 layers, at the embedding dimension, or width, of 768 used at 152M. The largest has 20 layers at a width of 1,024, with 16 attention heads and an MLP hidden dimension of 4,096. We train 3 seeds of each normalizing function at each of the four smaller sizes, and 8 at the largest.

Up to 227M, each normalizing function uses its learning rate and weight decay from Table 6. At 387M, we select the learning rate again, with one seed at each of three rates, namely, the rate given in Table 6, half of it, and a quarter of it. Half the rate gives the lowest perplexity for both normalizing functions. At that size, L1p trains at 2.4×10^{-3} and softmax at 4.8×10^{-3}, each with its weight decay unchanged. Table 17 reports every size, and Fig. 9 shows the difference between the two normalizing functions at each.

Table 17: Held-out perplexity of L1p and softmax at six model sizes, each the mean over the seeds given and the standard deviation across them. The comparison in Section 5.4 uses the model with 12 layers. The difference is the perplexity of L1p minus that of softmax, with its 95% confidence interval and the p-value of Welch's t-test.

Parameters	Layers	Width	Seeds	L1p	Softmax	Difference	p
76M	4	768	3	29.42 ± 0.09	28.77 ± 0.04	$+0.64$ [$+0.46$, $+0.83$]	2.0×10^{-3}
114M	8	768	3	24.89 ± 0.18	24.73 ± 0.04	$+0.16$ [-0.26, $+0.57$]	0.26
152M	12	768	24	23.20 ± 0.16	23.23 ± 0.17	-0.03 [-0.13, $+0.07$]	0.55
190M	16	768	3	22.21 ± 0.32	22.27 ± 0.12	-0.06 [-0.75, $+0.64$]	0.80
227M	20	768	3	21.64 ± 0.18	21.76 ± 0.18	-0.12 [-0.53, $+0.28$]	0.45
387M	20	1,024	8	19.42 ± 0.16	19.84 ± 0.21	-0.42 [-0.62, -0.22]	5.7×10^{-4}

The two normalizing functions differ in opposite directions at the smallest and the largest size. L1p has the higher perplexity at 76M, by 0.64, and the lower at 387M, by 0.42, with an interval wholly outside ± 0.20 in both cases. The two match at 152M. At 114M, 190M, and 227M, by contrast, the interval contains zero but extends beyond ± 0.20. These three sizes thus establish neither a match nor which normalizing function has lower perplexity.

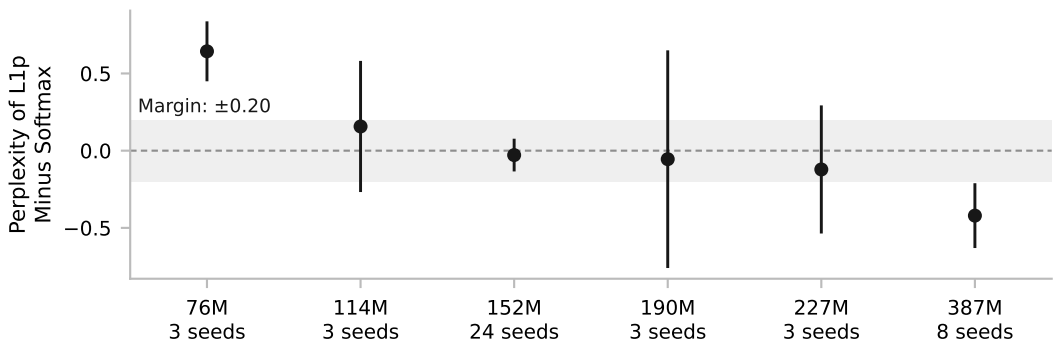

Figure 9: Held-out perplexity of L1p minus that of softmax at each model size, with its 95% confidence interval by Welch's t-test. A negative difference favors L1p. The grey band spans ± 0.20, and an interval that lies wholly within it establishes a match. Below each size is its number of seeds per normalizing function.

The largest size first had 3 seeds, like the other sizes except 152M. Over those seeds, the difference was -0.43, with an interval of $[-1.20, +0.34]$ that contains zero. Since this is the largest model in this paper, we added 5 seeds for both normalizing functions, and we added none at the other sizes. Because this decision followed the result over the first 3 seeds, we also report the 5 added seeds alone. Over them, the difference is -0.41, with an interval of $[-0.66, -0.17]$ and $p = 5.2 \times 10^{-3}$. L1p reaches the lower perplexity with each of the 8 seeds.

Two factors limit what the comparisons at these sizes establish. The first is the number of seeds. With 3 seeds at most sizes, the interval on the difference excludes zero only where the difference is large relative to the standard deviation across seeds. This standard deviation ranges from 0.04 to 0.32 over these sizes. The second is the learning rate, which we select again only at 387M. A rate chosen at 152M need not be the best rate at the other sizes. We check this at 227M, where halving the rate changes the perplexity of either normalizing function by at most 0.17, with one seed each. The smallest size, where L1p has the higher perplexity, has no such check.

D.4 Downstream benchmark tasks and scores

We evaluate 120 checkpoints from the comparison in Section 5.4 on downstream benchmarks. These are the final checkpoints of the 24 seeds of L1p, softmax, α-entmax, SSA, and SSA trained using the hyperparameters its authors reported. We use the Language Model Evaluation Harness (Gao et al., 2026), version 0.4.12, with the default settings of each task. Each checkpoint is scored on 20 tasks and on 4 measures of perplexity. These measures are the held-out perplexity on FineWeb and the perplexity on WikiText and on the two versions of LAMBADA in the harness. Of the 20 tasks, 7 are multiple choice and 2 are last-word prediction. The other 11 are long-context tasks from RULER (Hsieh et al., 2024) at a length of $1{,}500$ tokens, with 8 testing retrieval and 3 testing variable tracking or aggregation.

Models of this size score close to chance on some of these tasks, and a comparison on such a task carries little information. We call a task *above chance* when a single run of the normalizing function that scores highest on it can be distinguished from chance, and *at chance* otherwise. Specifically, the mean of that normalizing function must exceed chance by more than twice the noise of a single run. This noise combines the standard deviation across seeds with the sampling error of the task's own items. Of the 20 tasks, 12 are above chance. Table 18 reports every task and measure, with the 8 tasks at chance in its lowest block. Because a single run of even the best-scoring normalizing function cannot be distinguished from chance on these 8 tasks, Section 5 uses perplexity as the primary metric for language modeling.

Table 18: Downstream evaluation of five configurations at 152M parameters. Each entry gives the mean over 24 seeds and the standard deviation across them. SSA* denotes SSA trained using the hyperparameters reported by its authors. For a task labeled at chance, a single run of even the best-scoring normalizing function cannot be distinguished from chance. Table 19 reports the difference between L1p and softmax on every measure.

Measure	L1p	Softmax	α-entmax	SSA	SSA*
Perplexity					
FineWeb	23.20 ± 0.16	23.23 ± 0.17	22.78 ± 0.20	24.55 ± 0.22	26.91 ± 0.10
WikiText	35.50 ± 0.34	35.55 ± 0.36	34.67 ± 0.40	38.29 ± 0.46	43.34 ± 0.29
LAMBADA, OpenAI	38.6 ± 2.7	39.4 ± 3.1	36.6 ± 1.8	50.5 ± 3.9	69.6 ± 3.0
LAMBADA, standard	110.3 ± 11.4	104.7 ± 13.0	104.1 ± 11.6	151.1 ± 26.9	248.8 ± 27.7
Tasks above chance, accuracy in %					
lambada_openai	32.2 ± 1.1	32.1 ± 1.1	32.3 ± 0.9	28.6 ± 1.0	25.5 ± 0.6
lambada_standard	24.5 ± 1.1	25.8 ± 1.3	24.4 ± 1.0	21.7 ± 1.4	17.4 ± 0.8
arc_easy	40.8 ± 0.7	40.3 ± 0.6	41.0 ± 0.7	39.8 ± 0.6	38.7 ± 0.6
hellaswag	34.9 ± 0.3	34.9 ± 0.3	35.0 ± 0.4	32.8 ± 0.3	31.2 ± 0.2
piqa	66.1 ± 0.6	66.0 ± 0.6	66.4 ± 0.6	65.2 ± 0.6	64.2 ± 0.5
social_iqa	37.8 ± 0.7	37.9 ± 0.6	38.1 ± 0.6	37.2 ± 0.5	37.2 ± 0.6
niah_single_1	92.9 ± 10.2	83.3 ± 20.5	97.2 ± 3.6	82.0 ± 16.2	49.3 ± 28.1
niah_single_2	92.4 ± 7.3	91.8 ± 9.1	90.1 ± 13.7	75.1 ± 20.8	48.6 ± 24.2
niah_single_3	75.9 ± 18.9	77.1 ± 25.1	72.5 ± 24.7	58.9 ± 28.8	29.9 ± 18.2
niah_multikey_1	28.6 ± 3.3	33.8 ± 5.4	31.4 ± 4.3	25.2 ± 2.6	19.2 ± 1.6
niah_multiquery	15.1 ± 5.0	14.9 ± 4.5	16.7 ± 4.6	16.1 ± 3.4	14.9 ± 3.1
niah_multivalue	16.0 ± 4.7	14.7 ± 5.3	16.9 ± 4.3	17.1 ± 2.3	15.1 ± 1.8
Tasks at chance, accuracy in %					
arc_challenge	24.1 ± 0.7	24.3 ± 0.8	24.5 ± 0.8	23.8 ± 0.7	23.2 ± 0.7
openbookqa	29.0 ± 1.0	28.8 ± 0.8	29.0 ± 1.0	28.4 ± 0.9	28.3 ± 1.0
winogrande	51.0 ± 1.0	51.0 ± 1.2	50.9 ± 1.0	51.1 ± 1.0	51.0 ± 0.9
niah_multikey_2	1.3 ± 0.6	1.9 ± 1.0	1.2 ± 0.5	0.6 ± 0.2	0.3 ± 0.2
niah_multikey_3	0.2 ± 0.2	1.2 ± 1.1	1.2 ± 1.1	0.3 ± 0.3	0.0 ± 0.1
ruler_vt	0.5 ± 0.4	1.2 ± 1.5	1.6 ± 2.8	1.5 ± 2.2	3.0 ± 1.9
ruler_cwe	3.4 ± 1.8	2.8 ± 2.5	1.2 ± 1.1	0.5 ± 0.3	0.3 ± 0.2
ruler_fwe	3.0 ± 4.9	2.0 ± 3.0	2.5 ± 3.4	3.7 ± 3.1	5.4 ± 5.3

D.5 L1p AGAINST SOFTMAX ON DOWNSTREAM BENCHMARKS

To compare L1p with softmax directly, Table 19 gives the score of L1p minus that of softmax on every measure. Over 24 seeds, the interval on this difference excludes zero on 4 of the 12 tasks above chance. Softmax scores higher on lambada_standard, by 1.24 percentage points with $p = 1.0 \times 10^{-3}$, and on niah_multikey_1, by 5.12 points with $p = 3.0 \times 10^{-4}$. L1p scores higher on arc_easy, by 0.46 points with $p = 0.021$, and on niah_single_1, by 9.57 points with $p = 0.048$. On the other 8 tasks above chance, and on all 4 measures of perplexity, the interval contains zero. Among the 8 tasks at chance, the interval excludes zero on 3, each in favor of softmax and at accuracies below 2%.

At the 95% level, an interval excludes zero in about one of every twenty comparisons between two normalizing functions that do not differ. Among the 12 tasks above chance, chance alone would account for 0.6 intervals excluding zero in expectation, compared with the 4 we observe. This suggests that at least some of these 4 differences are real. They split evenly, with 2 in favor of each normalizing function. After a Holm correction over the 12 tasks, however, only the 2 in favor of softmax remain, on niah_multikey_1 (adjusted $p = 0.004$) and lambada_standard (adjusted $p = 0.011$). The benchmarks thus show no broad advantage for either normalizing function, but softmax scores higher on these two tasks.

Table 19: The score of L1p minus that of softmax on every measure in Table 18, with its 95% confidence interval and the p-value of Welch's t-test, over 24 seeds of each normalizing function. Differences in perplexity are in units of perplexity, and differences in accuracy are in percentage points. A negative difference in perplexity and a positive difference in accuracy favor L1p.

Measure	Difference	p
Perplexity		
FineWeb	-0.03 [-0.13, $+0.07$]	0.55
WikiText	-0.05 [-0.25, $+0.16$]	0.65
LAMBADA, OpenAI	-0.78 [-2.47, $+0.91$]	0.36
LAMBADA, standard	$+5.60$ [-1.50, $+12.70$]	0.12
Tasks above chance, accuracy in percentage points		
`lambada_openai`	$+0.04$ [-0.61, $+0.70$]	0.89
`lambada_standard`	-1.24 [-1.94, -0.53]	1.0×10^{-3}
`arc_easy`	$+0.46$ [$+0.07$, $+0.85$]	0.021
`hellaswag`	-0.04 [-0.23, $+0.15$]	0.67
`piqa`	$+0.01$ [-0.32, $+0.35$]	0.94
`social_iqa`	-0.13 [-0.50, $+0.24$]	0.48
`niah_single_1`	$+9.57$ [$+0.08$, $+19.07$]	0.048
`niah_single_2`	$+0.56$ [-4.22, $+5.33$]	0.81
`niah_single_3`	-1.23 [-14.16, $+11.70$]	0.85
`niah_multikey_1`	-5.12 [-7.72, -2.51]	3.0×10^{-4}
`niah_multiquery`	$+0.16$ [-2.62, $+2.94$]	0.91
`niah_multivalue`	$+1.39$ [-1.51, $+4.30$]	0.34
Tasks at chance, accuracy in percentage points		
`arc_challenge`	-0.17 [-0.62, $+0.28$]	0.45
`openbookqa`	$+0.20$ [-0.32, $+0.72$]	0.44
`winogrande`	$+0.02$ [-0.63, $+0.67$]	0.94
`niah_multikey_2`	-0.57 [-1.04, -0.11]	0.017
`niah_multikey_3`	-1.02 [-1.50, -0.55]	1.5×10^{-4}
`ruler_vt`	-0.68 [-1.33, -0.02]	0.043
`ruler_cwe`	$+0.63$ [-0.64, $+1.90$]	0.32
`ruler_fwe`	$+0.98$ [-1.41, $+3.37$]	0.41

D.6 Agreement between downstream benchmarks and perplexity

Section 5.4 compares the normalizing functions by perplexity. This subsection examines whether the downstream benchmarks order them in the same way. A measure *separates* two configurations when the 95% confidence interval on the difference between them excludes zero. For each pair of the five configurations in Table 18 and each task above chance, we check whether FineWeb perplexity and the task separate the pair. Where both separate it, we say that they *agree* if the configuration with the lower perplexity also scores higher on the task, and that they *disagree* otherwise. Over all ten pairs, perplexity and a task agree in 71 of the 73 comparisons in which both separate the pair.

Fig. 10 shows every outcome for three pairs that lie close in perplexity and three that lie far apart. The close pairs are those among L1p, softmax, and α-entmax, and the far pairs are those that each of them forms with SSA. The figure omits the four pairs that include SSA trained using the hyperparameters its authors reported. These lie more than 2 apart, and on each of them, 10 of the 12 tasks agree with perplexity and none disagrees. Both disagreements therefore appear in the figure. On `lambada_standard`, softmax scores higher than α-entmax, and on `niah_multivalue`, SSA scores higher than softmax, although in each case the other configuration has the lower perplexity.

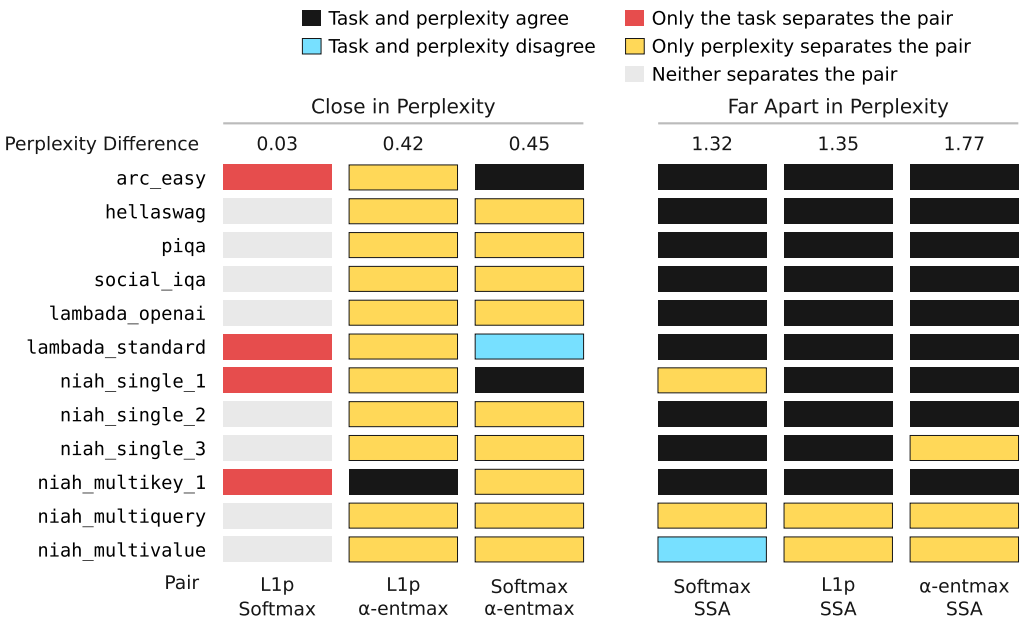

Figure 10: Agreement between each task above chance and FineWeb perplexity, on three pairs of configurations that lie close in perplexity and three that lie far apart. Each row is a task and each column a pair, with the difference between the mean perplexities of the pair above it. Each cell shows the outcome that the legend names. A measure separates a pair when the 95% confidence interval on the difference between its two configurations excludes zero. Where both separate a pair, the task and perplexity agree when the configuration with the lower perplexity also scores higher on the task.

On the two close pairs that include α-entmax, the tasks separate the pair far less often than perplexity does. Perplexity separates α-entmax from L1p and from softmax, which lie 0.42 and 0.45 apart, whereas only 1 and 3 of the 12 tasks do. By contrast, 9 or 10 of the 12 tasks separate each of the far pairs. In 28 of these 29 comparisons, the task agrees with perplexity. L1p and softmax are the only pair that perplexity does not separate, whereas 4 tasks separate them, 2 in favor of each normalizing function.

D.7 Reliability of single-run benchmark scores

The comparisons above use means over 24 seeds. A single run, by contrast, carries the full variation between seeds, which such a mean largely averages out. To measure how reliably a single run ranks two normalizing functions, we take the three pairs among L1p, softmax, and α-entmax, whose mean FineWeb perplexities differ by less than 1. For each pair, we compare every run of one normalizing function with every run of the other, 576 comparisons in all. We then count how often such a one-run comparison agrees with the 24-seed ranking, that is, ranks the two as the means of the same measure over 24 seeds do. A tie counts as half. Table 20 gives this rate for FineWeb and WikiText perplexity, and for the task above chance with the highest rate.

L1p and softmax match in perplexity, so their means give no clear ranking to recover. A one-run comparison by FineWeb or WikiText perplexity accordingly agrees with the 24-seed ranking in only about half of the cases. The two pairs that include α-entmax, by contrast, have a clear ranking, since perplexity separates each pair although its two normalizing functions lie within 1 of each other. For these two pairs, a one-run comparison by FineWeb or WikiText perplexity agrees in 95% or 96% of cases, whereas one on a task above chance agrees in at most 82%. Fig. 11 (a) shows these rates for every task above chance.

Table 20: How often a comparison of one run of each normalizing function agrees with the 24-seed ranking, that is, ranks the pair on a measure as the means of that measure over 24 seeds do. Each rate is over all 576 comparisons of one run of each, with a tie counted as half. The pairs are the three whose mean FineWeb perplexities differ by less than 1, and the first row gives that difference. The last row names the task above chance with the highest rate for each pair.

	L1p and Softmax	**L1p and α-entmax**	**Softmax and α-entmax**
Difference in mean perplexity	0.03	0.42	0.45
FineWeb perplexity	56%	95%	95%
WikiText perplexity	51%	96%	96%
Best task above chance	79%	70%	82%
Task	`niah_multikey_1`	`niah_multikey_1`	`niah_single_1`

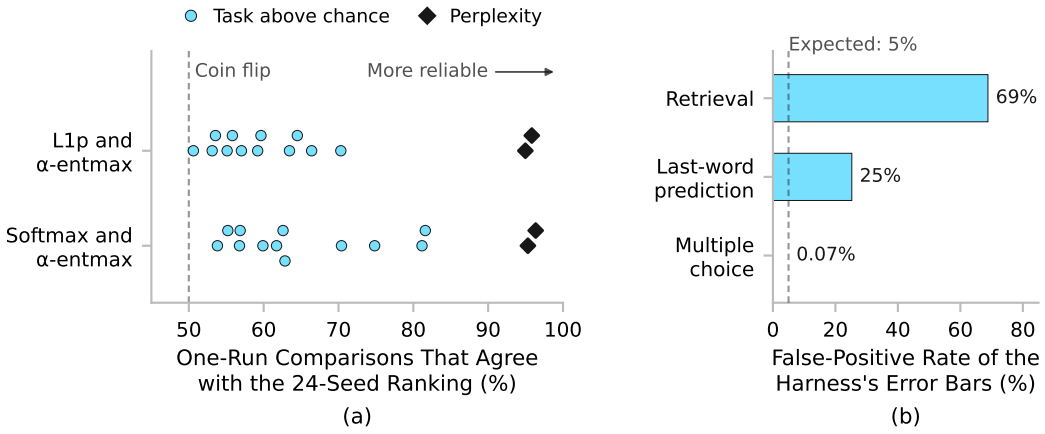

Figure 11: For the two close pairs that perplexity separates, perplexity from a single run recovers the 24-seed ranking far more often than any task above chance. The error bars that the evaluation harness reports misstate the variation between runs. (a) How often a comparison of one run of each normalizing function agrees with the 24-seed ranking, over all 576 such comparisons. The two pairs shown are the only ones that perplexity separates although they lie within 1 of each other. L1p and softmax, the third pair within 1 of each other, match in perplexity and leave no ranking to recover. Diamonds are perplexity on FineWeb and WikiText, and circles are the 12 tasks above chance. (b) How often these error bars call two runs of the same configuration significantly different, pooled over the tasks above chance of each kind. The dashed line marks the 5% expected if the error bars captured all the variation between runs.

The evaluation harness also reports an error bar with each score. This error bar reflects only the sampling of the task's items, and not the variation between runs. We call two runs *significantly different* when their scores differ by more than 1.96 times their combined error bar. For two runs of the same configuration, which differ only in their seed, a significant difference is a false positive. If the error bars captured all the variation between runs, false positives would occur in about 5% of these pairs of runs. On the tasks above chance, they occur in 69% of pairs on retrieval and 25% on last-word prediction, but in only 0.07% on multiple choice, as Fig. 11 (b) shows. The error bars thus understate the variation between runs on retrieval and last-word prediction, and overstate it on multiple choice. Judged against its error bar, a retrieval score from a single run is accordingly weak evidence about the normalizing function.

E Design Choices and Ablations

This appendix reports the experiments that support each design choice in Table 3. Each subsection addresses one aspect of the design. Table 21 gives, for each alternative mechanism, for signed softmax, and for each variant of L1p, the number of runs that solve the multi-hop reasoning task at 4 hops. A run solves the task at a given number of hops when its rescaled accuracy on questions with that number of hops reaches 50, as Appendix C describes. A signed layer is a layer that uses the signed form of L1p. Because some of these experiments change more than one component of L1p, each subsection states what its experiments change.

Table 21: The number of runs, out of 16, in which each alternative mechanism, signed softmax, and each variant of L1p solves the multi-hop reasoning task at 4 hops, with the p-value of Fisher's exact test against L1p. Every row uses the same 16 seeds, the first 16 of the 32 that the comparison in Section 5.3 uses. Each alternative mechanism and signed softmax runs at its own tuned learning rate, and each variant of L1p at the learning rate of L1p. Boldface marks the largest count.

Configuration	**Runs that Solve 4 Hops, of 16**	**p-value Against L1p**
L1p	9	—
Alternative mechanisms and signed softmax		
Exponential form	2	0.023
Magnitude-invariant softmax	0	8.2×10^{-4}
NormSoftmax (Jiang et al., 2023)	0	8.2×10^{-4}
Signed softmax	1	5.9×10^{-3}
Variants of L1p		
Exponent held fixed at 4	**11**	0.72
Exponent held fixed at 1	0	8.2×10^{-4}
Signed form removed	0	8.2×10^{-4}
Signed form in the first 6 layers	1	5.9×10^{-3}

E.1 Alternative mechanisms for controlling sharpness

L1p sharpens attention through a power of the magnitudes of the query–key products, with a learnable exponent. We compare it with the three alternative mechanisms for controlling sharpness that Appendix A.1 defines, and with signed softmax, the control introduced in Section 5.1. Table 22 lists the properties of L1p that each of these four shares. Each runs at its own tuned learning rate, given in Table 6.

As Table 22 shows, the exponential form differs from L1p only in computing its scores from an exponential of the magnitudes rather than from a power. It divides the magnitudes of the products by their root mean square over the row, multiplies them by a learnable inverse temperature, and applies softmax. The signs are restored in the same layers as in L1p. The exponential form thus isolates whether the attention scores should come from a power of the magnitudes or from an exponential of them. The exponential form solves the reasoning task at 4 hops in 2 of its 16 runs, against 9 for L1p, and Fisher's exact test gives $p = 0.023$. At its learning rate, 5 of its runs remain close to the accuracy of guessing even at 1 hop. The two forms also differ in what sets the ratio between two attention scores. In L1p, this ratio depends only on the two query–key products that produce the scores, as Section 4 shows. In the exponential form, the ratio also depends on the rest of the row, through the root mean square by which the magnitudes are divided.

Table 22: The properties of L1p that each alternative mechanism and signed softmax share. A mechanism is magnitude-invariant when its attention scores are unchanged under a common positive rescaling of the query–key products. Its sharpness is learnable when a learned parameter sets it, namely, the exponent of L1p or an inverse temperature. NormSoftmax is magnitude-invariant only while the standard deviation of the products over the row stays below its threshold.

Mechanism	Signed Scores	Magnitude-Invariant	Learnable Sharpness	Form
L1p	Last 6 layers	Yes	Yes	Power
Exponential form	Last 6 layers	Yes	Yes	Exponential
Magnitude-invariant softmax	No	Yes	Yes	Exponential
NormSoftmax	No	Partly	No	Exponential
Signed softmax	Every layer	No	No	Exponential

The other two alternative mechanisms and signed softmax each lack at least one property of L1p. Magnitude-invariant softmax lacks signed scores; NormSoftmax lacks learnable sharpness and is magnitude-invariant only in part; and signed softmax lacks both magnitude invariance and learnable sharpness. Magnitude-invariant softmax and NormSoftmax solve the reasoning task at 4 hops in none of their 16 runs, and signed softmax solves it in 1. Magnitude-invariant softmax also differs from L1p in its gradient, which can change the overall magnitude of the products without changing the scores, as Appendix A.1 describes. Appendix B.6 compares the sharpness of every normalizing function with the reasoning depth it reaches.

E.2 Learned or fixed exponent

The exponent of L1p is learned during training rather than held fixed. Holding it fixed at 4 does not significantly change reasoning depth. On the 16 seeds of Table 21, the variant with the exponent fixed at 4 solves the reasoning task at 4 hops in 11 runs, against 9 for L1p ($p = 0.72$). It solves the task at 2 hops in fewer runs, 12 against 16 ($p = 0.10$), but at 8 hops in more, 7 against 2 ($p = 0.11$), and at 16 hops in 2, against none ($p = 0.48$). Each p-value is from Fisher's exact test.

This variant also moves the exponent from its initial value of 6 to 4. To separate these two changes, a second comparison changes only whether the exponent is learned. On 8 seeds, we train one configuration with the exponent fixed at 4 and one with a learnable exponent initialized at 4, both with 6 signed layers. The learning rate is 5×10^{-4} on the reasoning task, as for L1p, and 6×10^{-4} on language modeling. On the reasoning task, 7 of the 8 runs solve the task with the fixed exponent and 6 with the learnable one, with $p = 1.0$. At 8 hops, 4 of the 8 runs solve the task with the fixed exponent and 2 with the learnable one ($p = 0.61$). On language modeling, the fixed exponent raises perplexity by 0.29, with a 95% confidence interval of $[+0.19, +0.39]$ and $p = 2.7 \times 10^{-5}$. It gives the higher perplexity on every one of the 8 seeds. The case for a learnable exponent, which adds one parameter per layer, rests on language modeling.

E.3 Value of the exponent

An exponent above 1 is what sharpens attention in L1p. Holding the exponent fixed at 1 removes this sharpening, and no run solves the reasoning task at 4 hops, against 11 of 16 with the exponent fixed at 4, with $p = 6.8 \times 10^{-5}$. At an exponent of 1, the layers with the signed form compute ℓ_1 normalization, and the other layers divide the magnitude of each product by the ℓ_1 norm of the row. The ℓ_1 normalization in Table 2 differs from this variant in using the signed form in all 12 layers and in running at its own learning rate.

The exponent is learned, but it still needs an initial value. We sweep 1, 2, 4, 8, and 16 on both tasks, and Table 23 reports the result. An initial value of 4 yields the most runs that solve the reasoning task, 6 of 8, whereas an initial value of 8 yields the lowest perplexity, 23.77. After this sweep, we also train L1p with the exponent initialized at 6, between the two, with the same settings on 8 different seeds. With this value, 6 of 8 runs solve the reasoning task, as many as with 4, and the perplexity is 23.86, within 0.10 of the lowest. We accordingly initialize the exponent at 6.

Table 23: The initial value of the exponent, swept on both tasks with 6 signed layers and 8 seeds per value. The reasoning task uses a learning rate of 5×10^{-4}, and language modeling one of 6×10^{-4}. Perplexity is the mean over the seeds and the standard deviation across them. The value marked † was trained after the sweep, with the same settings on 8 different seeds. Boldface marks the best value on each task.

Initial Value	Runs that Solve 4 Hops, of 8	Perplexity
1	3	24.77 ± 0.08
2	3	24.45 ± 0.08
4	**6**	24.10 ± 0.09
$6^{\dagger}$	**6**	23.86 ± 0.08
8	2	$\mathbf{23.77 \pm 0.06}$
16	0	24.08 ± 0.08

E.4 Whether and where to use the signed form

Two variants change only the signed form of L1p. The first removes the signed form, so that all 12 layers use the unsigned form, with the exponent initialized at 6 as in L1p. None of its runs solves the reasoning task at 4 hops, against 9 for L1p, and Fisher's exact test gives $p = 8.2 \times 10^{-4}$. The second variant moves the signed form from the last 6 layers to the first 6. This variant solves the task in 1 run, with $p = 5.9 \times 10^{-3}$. Reasoning depth thus depends both on the signed form and on where it is placed.

E.5 Number of signed layers

The signed form can be used in any number of layers, and the number of signed layers affects both reasoning depth and perplexity. We train L1p with the signed form in the last 0, 1, 2, 3, 6, or 12 of its 12 layers, with the exponent initialized at 4. The sweep uses 24 seeds on the reasoning task, at a learning rate of 5×10^{-4}, and 8 on language modeling, at a learning rate of 6×10^{-4}. The reference for perplexity is softmax, trained with 12 seeds at the same learning rate. Table 24 reports every setting, and Fig. 12 shows both tasks together.

Table 24: Reasoning depth and perplexity against the number of signed layers, with the exponent initialized at 4. Reasoning depth is over 24 seeds. The perplexity columns give the mean perplexity over 8 seeds minus that of the setting without signed layers and minus that of softmax, which is trained with 12 seeds at the same learning rate of 6×10^{-4}. The initial exponent and the language modeling learning rate differ from those of L1p, so the table is not comparable with the results in Sections 5.3 and 5.4. Boldface marks the largest count.

		Perplexity Relative to	
Signed Layers	Runs that Solve 4 Hops, of 24	No Signed Layers	Softmax
None	1	—	-0.14
Last 1	0	$+0.04$	-0.10
Last 2	2	$+0.05$	-0.09
Last 3	7	$+0.10$	-0.04
Last 6	12	$+0.24$	$+0.10$
All 12	**20**	$+0.96$	$+0.82$

Reasoning depth increases with the number of signed layers. With no signed layers, 1 of 24 runs solves the task at 4 hops, and with the signed form in all 12 layers, 20 runs do. The Cochran–Armitage test across the five settings with signed layers gives $z = 6.80$ and $p = 1.1 \times 10^{-11}$. A single signed layer does not increase reasoning depth on its own. No run solves the task with 1 signed layer, against 1 with none, and Fisher's exact test gives $p = 1.0$. Perplexity rises with the number of signed layers, by 0.04 at 1 signed layer and by 0.96 at 12.

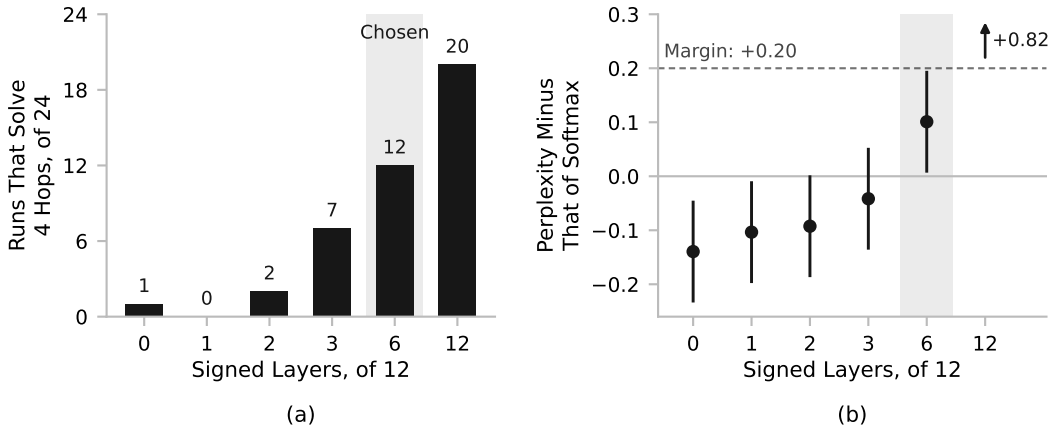

Figure 12: Reasoning depth and perplexity against the number of signed layers, with the exponent initialized at 4. (a) Runs that solve the reasoning task at 4 hops, of 24. (b) Perplexity minus that of softmax, with the 99% interval of the one-sided test in the selection rule. A setting meets the condition of the rule on perplexity when its interval lies below the margin of $+0.20$. The setting with all 12 signed layers lies off the scale, at $+0.82$. The grey band marks the chosen setting, 6 signed layers.

We choose the number of signed layers with a rule that weighs both tasks. A setting qualifies if it meets two conditions. First, its mean rescaled accuracy at 4 hops must exceed that of the setting without signed layers by more than the standard deviation across seeds, pooled over the two settings. Second, its perplexity must be shown to lie at most 0.20 above that of softmax, by a one-sided t-test at the 0.005 level. This test does not use the standard deviations of the settings it compares. It uses 0.070, the standard deviation across seeds pooled over the 12 softmax runs of the reference and 12 further runs of the setting with 6 signed layers, all trained on the same seeds. When several settings qualify, the rule keeps those that no other qualifying setting outperforms on both tasks, and among them the one with the fewest signed layers. Only the setting with 6 signed layers meets both conditions, so the rule selects it. The settings with 1, 2, and 3 signed layers fail the first condition, and the setting with all 12 fails the second, at 0.82 above softmax.

This sweep initializes the exponent at 4 and trains language models at a learning rate of 6×10^{-4}. The sweep consequently selects the number of signed layers under hyperparameters that differ from those of L1p. On the reasoning task, two of its settings have also been trained with the exponent initialized at 6, on the same 16 seeds as Table 21. The setting without signed layers solves the task in none of its 16 runs, as Appendix E.4 reports, and the setting with 6, which is L1p itself, solves it in 9.

F Efficiency

This appendix compares the efficiency of L1p and softmax, in training and in generation. All runs in this paper compute attention with one shared implementation, composed of standard PyTorch operations. We call it the *unfused implementation*. Since every normalizing function uses it, a comparison between L1p and softmax on this implementation differs only in the normalizing function. For softmax, PyTorch also provides a fused attention kernel, through `scaled_dot_product_attention`, which none of the runs in this paper uses. L1p does not yet have a fused kernel.

F.1 Wall-clock time of the training runs

All L1p and softmax runs of the main comparisons trained on the same two machines. On language modeling, a run of L1p took 2.35 hours on average and a run of softmax 1.98 hours, a ratio of 1.19. On the reasoning task, the corresponding averages were 8.44 and 7.97 hours, a ratio of 1.06.

On each task, this ratio differs by less than 0.001 between the two machines. These times cover complete runs. They include evaluation and the measurements of attention entropy described in Appendix B.1, which also compute the normalizing function. For L1p, these measurements also record additional statistics, so the times of L1p include work that the softmax runs do not perform.

F.2 Time per training step

To exclude evaluation and the measurements of attention entropy, we time individual training steps on one NVIDIA H100 80GB accelerator, with the 152M language model at its random initialization. Each step processes $16{,}384$ tokens, so the batch size halves whenever the context length doubles. Each time is the median over 10 steps, following 3 warm-up steps. We time the steps both with the model compiled by `torch.compile`, as in the runs of this paper, and without compilation. Table 25 reports the results, together with those of softmax computed by the fused kernel.

Table 25: Time per training step on one accelerator, in milliseconds, for the 152M language model at its random initialization, with $16{,}384$ tokens per step. Each entry is the median over 10 steps. The first two rows use the unfused implementation of attention, and the third row uses the fused kernel that PyTorch provides for softmax. A dash marks a step that exceeds the memory of the accelerator in this benchmark. The last row gives the time of L1p relative to that of softmax on the unfused implementation, computed from the unrounded times.

	Compiled			**Uncompiled**		
Context Length	**512**	**1,024**	**2,048**	**512**	**1,024**	**2,048**
Softmax	54.8	66.4	93.1	130.7	178.7	274.9
L1p	56.9	69.5	106.6	200.7	317.0	—
Softmax, fused kernel	45.0	46.2	48.7	80.3	81.5	84.0
L1p relative to softmax	1.04	1.05	1.14	1.53	1.77	—

With compilation, a step of L1p takes 1.04 and 1.05 times as long as a step of softmax at context lengths of 512 and $1{,}024$. The ratio rises to 1.14 at $2{,}048$, the context length used to train the language models of this paper. Without compilation, the ratio is 1.53 at 512 and 1.77 at $1{,}024$. At $1{,}024$, compilation reduces the ratio from 1.77 to 1.05. At $2{,}048$, the uncompiled step of L1p exceeds the memory of the accelerator in this benchmark.

With the fused kernel, a compiled step of softmax runs 1.91 times as fast as with the unfused implementation at $2{,}048$. The step time of the fused kernel also changes little with the context length. Relative to softmax with the fused kernel, a compiled step of L1p on the unfused implementation takes 2.19 times as long at $2{,}048$. This ratio is, up to rounding, the product of two ratios, namely 1.14 between the two normalizing functions on the unfused implementation and 1.91 between the unfused implementation and the fused kernel.

F.3 GENERATION

Table 26: Generation with the 152M language model on one accelerator, at a batch size of 8 and without compilation, in one of three separate executions of the benchmark. The first column gives the time to process a prompt of 2,048 tokens, and the second the rate at which 128 new tokens are then generated. The last row gives the time and the rate of L1p relative to those of softmax on the unfused implementation.

	Prompt Time, ms	**New Tokens per Second**
Softmax	105.1	1,273
L1p	240.3	952
Softmax, fused kernel	25.7	1,502
L1p relative to softmax	2.29	0.75

We also time generation from a prompt of 2,048 tokens, followed by 128 new tokens generated with a cache of keys and values, at a batch size of 8 and without compilation. We ran this benchmark in three separate executions, and Table 26 reports the times measured in one of them. L1p takes 2.29 times as long as softmax to process the prompt, and this ratio lies between 2.28 and 2.29 in all three executions. In the execution shown, L1p generates new tokens at 0.75 times the rate of softmax. This ratio varies between executions, from 0.75 to 0.82. Between executions, the rate of softmax varies by up to a factor of 1.07, and that of L1p by up to 1.04. We did not measure generation with compilation. Since compilation reduces the ratio between the two normalizing functions for a training step, the ratios in Table 26 may differ under compilation.

G RELATED WORK

This appendix surveys prior work on reasoning depth, on the dependence of softmax on the overall magnitude of its inputs, and on alternatives to softmax. It then places L1p among these alternatives and reviews the methodology for evaluating them. Table 1 summarizes the properties of L1p and of the established normalizing functions compared in Section 5, and Table 27 repeats its rows and adds three more normalizing functions that this appendix discusses.

G.1 PRIOR WORK ON REASONING DEPTH

Prior work has studied the reasoning depth of Transformers from two directions, one theoretical and one empirical. On the theoretical side, Sanford et al. (2024) analyzed k-hop induction, in which a model follows a chain of pointers k times. They proved that a Transformer with $\lfloor \log_2 k \rfloor + 2$ layers can solve this task with a constant embedding dimension and a single head per layer. They also proved a matching lower bound on the number of layers. This lower bound assumes the one-versus-two-cycle conjecture in the massively parallel computation model and holds only within a particular range of parameters. Together, the two bounds suggest that, when the width and the number of heads are small relative to the number of hops, the number of layers limits the number of hops that a Transformer can compose.

Also on the theoretical side, Peng et al. (2024) examined how much information a single attention layer can transmit. They reduced the composition of two functions to a one-way communication game among three parties. Each of two parties holds one of the functions and may send messages only to a third party holding the input to which the composition is applied. Using this reduction, they proved that a single layer with h heads, embedding dimension D, and b-bit precision cannot compose the two functions correctly when $h(D+1)b < m \log m$, where m is the size of the domain of each function.

Their theorem is stated for softmax. The proof, however, uses only the fact that the numerator and the denominator of the output of a head are sums of terms that each depend on a single attended token. The output of a head with L1p has the same form, so the proof applies to L1p as well. This application of their proof to L1p is our own. The proof does not cover normalizing functions in

which each term depends on every token of the row. In sparsemax, for instance, every term depends on a threshold that is computed from all query–key products of the row.

On the empirical side, Curth et al. (2026) studied kinship questions whose answers require following a chain of relations, reporting accuracy separately for each length of the chain. They used the logit lens to locate the layer at which the answer becomes decodable, and causal patching to locate the layer at which the information from each relation reaches the final token. They found that models generally used more layers to integrate information across tokens as the chain grew longer. The authors qualified this finding, noting that in pretrained models the supporting evidence was limited in quantity, mixed in direction, and not strongly consistent. In fine-tuned models, the evidence was clearer and more consistent, especially under full fine-tuning.

Allen-Zhu (2025) introduced a family of controlled synthetic pretraining tasks, each isolating a single skill of a language model. The multi-hop reasoning task in this paper is adapted from one of these tasks. That work compared positional encodings, including rotary encodings and no positional encoding, and introduced lightweight residual layers that combine the representations of neighboring tokens. For each resulting configuration, the authors reported how many hops the model could chain. The same work stated that a failure under a fixed training budget reflects the speed at which a skill is acquired, rather than whether the skill can be acquired at all. The results of this paper are likewise obtained under a fixed training budget, and they measure the reasoning depth reached within this budget.

Another empirical study, by Duvvuri et al. (2026), compared several attention mechanisms on reasoning benchmarks at scale and on smaller synthetic tasks of relation composition. Some of these mechanisms exchanged information across heads, but all of them computed their attention scores with softmax.

Across these studies, the normalizing function was never varied on its own. The studies varied the number of layers, the positional encoding, the arrangement of heads, and the addition of residual layers, with each study designed to isolate one of these factors. When these studies replaced softmax, they changed more of the attention layer than its normalizing function. For instance, Allen-Zhu (2025) also compared linear-attention and state-space models, and Sanford et al. (2024) analyzed kernel attention.

Hoffmann et al. (2024) replaced only the normalizing function, but their study concerned optimization rather than reasoning depth. They considered two-step compositional tasks on which the accuracy of Transformers plateaued for hundreds of epochs before rising abruptly. They attributed the plateau to softmax, whose derivative is close to zero when attention is close to uniform. In their measurements, the gradients of the query and key projections were half to one order of magnitude smaller than those of the value projection.

In the same study, replacing softmax with NormSoftmax (Jiang et al., 2023) shortened the plateau considerably. NormSoftmax divides the query–key products by the smaller of their standard deviation and a threshold. Under softmax, the gradients were small but not zero, and with sufficiently long training, the abrupt rise in accuracy occurred for every seed that the authors tried. The plateau thus delayed the acquisition of the skill rather than preventing it. That study did not relate accuracy to the number of hops. To the best of our knowledge, the effect of the normalizing function on reasoning depth, with the rest of the attention layer held fixed, has not yet been studied.

G.2 Growth of the query–key products and its remedies

As Section 3 describes, the sharpness of softmax changes with the overall magnitude of the query–key products, a magnitude that can grow during training (Wortsman et al., 2024; Dehghani et al., 2023). Wortsman et al. (2024) attributed the growth to an increase in the norms of the query and key vectors, rather than to closer alignment between queries and keys. Zhai et al. (2023) characterized the collapse of attention entropy that can accompany this growth and proposed a spectral reparameterization of the weights to prevent this collapse.

Several remedies constrain the query–key products. The most common remedy normalizes the query and key vectors before their dot product, either to unit length with a learned scale (Henry et al., 2020) or with layer normalization (Dehghani et al., 2023). In both cases, learned quantities bound the products. A more recent approach limits how much the products can change between consecutive steps

of training (Anson & Aitchison, 2025). Each of these remedies acts through a mechanism outside the normalizing function. Because Section 5 varies only the normalizing function, its comparisons do not combine softmax with any of these remedies. By contrast, the scores of L1p do not depend on the overall magnitude of the products, a property that Section 4 calls *magnitude invariance*.

G.3 Theoretical properties of normalizing functions

Mudarisov et al. (2025) bounded the change in the attention scores that a small change in the query–key products can cause. Their bound concerns normalizing functions that apply a positive function to each product and divide each resulting value by the sum of these values over the row. The unsigned form of L1p has this structure, whereas the signed form does not, since the numerator of the signed form carries the sign of the product. The term of their bound that depends on the products grows without limit as the smallest value of this function over the row approaches zero. For softmax, this term is finite, since the exponential is positive everywhere. For the unsigned form of L1p, the function is a power of the magnitude of the product, and this power vanishes when the product is zero, so this term becomes infinite. Their bound assumes a smooth and strictly positive function, a condition that a power of the magnitude violates at zero. This evaluation of their bound for L1p is our own.

For normalizing functions of this form, the divergence of the bound at a zero product is inherent to magnitude invariance. Magnitude invariance requires the function applied to each product to be homogeneous, and a homogeneous function that increases with the magnitude of its input is zero at zero. A direct computation shows, however, that for $p \geq 2$ the Jacobian of the scores of L1p with respect to the products diverges only in a narrower case. Under magnitude invariance, the Jacobian scales inversely with the overall magnitude of the products, and for $p \geq 2$ its Frobenius norm is at most $\sqrt{2}\,p$ divided by the p-norm of the row of products. This norm can therefore grow without limit only when the whole row approaches zero, and not when a single product does. Qin et al. (2022) identified this case as a source of instability for normalized attention with non-negative scores. The normalization that Section 4.2 describes keeps the scores in such rows close to the scores that the ideal form of L1p assigns. This normalization leaves the gradient with respect to the products unchanged, however, so this gradient still grows as the whole row approaches zero.

Veličković et al. (2025) proved that softmax attention learned on a finite vocabulary must disperse toward a uniform distribution as the number of attended items grows at test time. The result holds even for a task as simple as retrieving the maximum. It assumes continuous feedforward and attention blocks, and the authors noted that unnormalized, hard, and discontinuous attention fall outside its scope. Mudarisov et al. (2025) generalized the result to normalizing functions built on a smooth function that is bounded below by a positive constant. Under bounded query–key products, such normalizing functions, softmax included, confine the attention scores to $\Theta(1/n)$ over n attended tokens.

L1p does not satisfy the conditions of this generalization, since a power of the magnitude is zero at zero. The confinement to $\Theta(1/n)$ can then fail. If all products of a row are zero except one, the token with the non-zero product receives a score of magnitude one for every n. Whether dispersion constrains L1p remains open. Our experiments, which hold the context length fixed, do not address this question.

A separate line of work adapts attention to the context length directly. Nakanishi (2025) rescaled the query–key products by a function of the sequence length, and Anson et al. (2025) combined a position-dependent transform of the products with rotary embeddings. These methods target the length of the context, whereas L1p removes the dependence on the overall magnitude of the products at a fixed length.

G.4 Sparse normalizing functions

The alternatives to softmax surveyed in this subsection and the next two each relax at least one of three properties of softmax. Softmax assigns every token a positive score, makes the output of a head a convex combination of the value vectors, and derives its scores from an exponential of the products. Sparse normalizing functions relax the first of these properties while keeping the scores on the probability simplex.

Sparsemax (Martins & Astudillo, 2016) projects the query–key products onto the probability simplex, assigning a score of exactly zero to every token whose product falls below a threshold. The α-entmax family (Peters et al., 2019) includes softmax and sparsemax as special cases, with α setting the degree of sparsity. Subsequent work learned α for each attention head (Correia et al., 2019), made the computation efficient on hardware (Gonçalves et al., 2025), and applied α-entmax to long-context generalization (Vasylenko et al., 2026). The evaluations of these functions spanned natural language inference, multi-label classification, machine translation, language-model pretraining, retrieval, and long-context generalization.

The closed form of α-entmax raises a shifted and rectified product to the power $1/(\alpha - 1)$. This power exceeds one for $1 < \alpha < 2$ and equals one at $\alpha = 2$, where α-entmax reduces to sparsemax. For $1 < \alpha < 2$, α-entmax sharpens attention in two ways, by discarding the tokens below the threshold and by raising the shifted products of the remaining tokens to a power above one. Tables 1 and 27 mark α-entmax in both columns concerning the exponent, since this power can exceed one and α can be learned. Because the power acts on a shifted product rather than on the product itself, the scores of α-entmax are not magnitude-invariant.

Sparse normalizing functions retain the convex combination and the dependence on the overall magnitude of the products. The convex combination remains because the scores are non-negative and sum to one. The dependence on the overall magnitude arises from the threshold, which is set by the constraint that the scores sum to one. As a result, rescaling the products can change which tokens receive a non-zero score. Naim et al. (2026) reported that neither sparsemax nor α-entmax improved over softmax on their in-context learning tasks under distribution shift. Section 5 compares both functions with L1p.

G.5 Signed and unnormalized attention scores

The convex combination can be relaxed in two stages. The first stage permits negative scores, allowing a head to subtract a value vector as well as add it. Cog Attention (Lv et al., 2024) exponentiates the magnitude of each product, restores the sign of the product, and divides by the sum of the magnitudes of the results. The exponential makes the scores of Cog Attention depend on the overall magnitude of the products.

Lv et al. (2024) initially applied Cog Attention in every layer and reported that convergence slowed. Their final models accordingly kept softmax in the first and last layers. The authors attributed the slower convergence to the signs of the scores being uninformative early in training. They also tried a cubic function in place of the exponential. Under their normalization, the resulting variant of Cog Attention equals the signed form of L1p with the exponent fixed at 3. They retained the exponential for the convenience of its gradient and reported no results for the cubic function. Signed softmax, the control in our experiments, computes the scores of Cog Attention in every layer, including the first and the last, as Appendix A.1 describes.

Two other designs also produce signed scores. The Differential Transformer (Ye et al., 2025) subtracts one softmax map from another. Its two maps use separate query and key projections, the subtracted map carries a learned weight, and the output of each head is normalized. Because it changes more of the attention layer than the normalizing function, Section 5 does not include it. ZeroS (Lu et al., 2025) removes the zero-order term of the Taylor expansion of softmax, leaving gated residuals that sum to zero rather than to one. PolaFormer (Meng et al., 2025), a linear attention for vision Transformers, by contrast keeps its scores non-negative. It separates queries and keys into positive and negative parts and recombines the interactions of opposite sign through learned coefficients rather than by subtraction. Its authors adopted this design to avoid an instability that they attributed to subtraction. PolaFormer learns an exponent, constrained above one, that acts on each channel of the kernel feature map rather than on the query–key products.

The second stage removes the convex combination altogether. Richter & Wattenhofer (2020) observed that a convex combination confines the output of a head to the convex hull of its value vectors. They showed that this confinement limits expressivity, since no convex combination can represent exclusive-or, whereas a normalized weighting can. They proposed Normalized Attention Pooling, which replaces softmax with the z-score of the query–key products, followed by a learnable gain and bias. The resulting scores can be negative and do not change when the products are rescaled or shifted. Because the scores are linear in the z-score, Normalized Attention Pooling does not sharpen

attention. In the experiments of Richter & Wattenhofer (2020), simplex-constrained models trained successfully over a narrower range of hyperparameters than models without that constraint.

The evidence for Normalized Attention Pooling came from synthetic set and sequence tasks, over more than $25{,}000$ trained models with 5 seeds per configuration. The comparison covered six architectures, namely the normalized and unnormalized variants of the method, a BERT-style encoder, a modified encoder that keeps softmax, and sum- and max-pooling. The fully unnormalized variant performed worse than the normalized one. No language modeling experiments were reported.

G.6 Powers in place of the exponential

Several normalizing functions replace the exponential with a power. The ℓ_1 normalization in our comparisons derives from Tensorformer (Tian et al., 2025), a method designed for point-cloud surface reconstruction. In Tensorformer, each attention weight is a matrix that a small network computes from the difference between two point features. The signed entries of this matrix are divided by the sum of their magnitudes along the channel dimension. This division makes the weights magnitude-invariant and corresponds to an exponent fixed at one. Our ℓ_1 normalization applies the same division to each row of query–key products.

PowerSoftmax (Zimerman et al., 2026), designed for secure inference over encrypted data, raises the products to an even power and divides by the sum of these powers. The even power makes the scores of PowerSoftmax non-negative. PowerSoftmax thus computes the scores of the unsigned form of L1p with a fixed even exponent. Before applying the power, PowerSoftmax also divides each row of products by its largest magnitude, a division that leaves the scores unchanged because PowerSoftmax is magnitude-invariant. L1p likewise divides each row before applying the power, using the root mean square of the row in place of the largest magnitude, as Section 4 describes. Powers have also appeared in a sketched polynomial kernel (Kacham et al., 2024) and in a tunable state expansion for training on long contexts (Gelada et al., 2025). Kacham et al. (2024) also reported that polynomial attention of high degree can replace softmax without a loss in model quality. Neither work treated the exponent as the object of study.

Saratchandran et al. (2024) argued that the effectiveness of softmax stems from an implicit regularization. Softmax bounds the Frobenius norm of the attention matrix by a quantity that grows as the square root of the sequence length. A cubic function divided by a constant reproduces this rate on average over random parameters, and hence at initialization. Attention with this cubic function matched the accuracy of softmax attention in vision Transformers, on Long Range Arena, and in physics-informed networks. The authors reported that models with exponents above 3 were difficult to train. They separately reported that models with polynomials of order above 6 trained poorly. They attributed both difficulties to a high power mapping values near zero much closer to zero.

Scaled Signed Averaging (Naim et al., 2026) also replaces the exponential with a learnable function whose exponent carries the sign of the query–key product. Because the sign enters through the exponent rather than through the numerator, the scores remain non-negative and sum to one. Naim et al. (2026) showed that, as the positive products grow without bound, Scaled Signed Averaging approaches the behavior of a normalized power. They reported improvements on in-context learning under distribution shift, in language modeling, and in grammatical probing of encoder models. Their evaluation centered on two in-context learning tasks, inferring affine functions and judging the quantifiers *every* and *some*.

Two names that could describe L1p already refer to other methods. The name scale-invariant attention is used for invariance to the context length (Anson et al., 2025), and the name power attention for a state expansion in linear attention (Gelada et al., 2025). To avoid confusion with either, we use the term *magnitude-invariant*, as defined in Section 4.

G.7 Position of L1p among these normalizing functions

The normalizing functions surveyed in the preceding subsections show that each of the three properties of softmax can be relaxed in models that train successfully. Their evaluations covered point-cloud reconstruction, encrypted inference, vision, efficient attention, long-context retrieval, in-context learning, and language modeling at scale. None of these evaluations measured reasoning

depth. Conversely, the studies of reasoning depth discussed above did not vary the normalizing function on its own.

Table 27: Properties of the main normalizing functions discussed in this appendix. *Signed* means that attention scores may be negative, and *magnitude-invariant* means that they are unchanged when all query–key products are rescaled by a common positive factor. The last two columns indicate whether a power above one is used and whether it is learned. Table 1 lists the seven of them compared in Section 5.

Normalizing Function	**Signed**	**Magnitude-Invariant**	$p > 1$	p **Learnable**
Softmax (2017)	✗	✗	✗	✗
Sparsemax (2016)	✗	✗	✗	✗
α-entmax (2019; 2019)	✗	✗	✓	✓
Normalized Attention Pooling (2020)	✓	✓	✗	✗
ℓ_1 normalization (2025)	✓	✓	✗	✗
Cog Attention (2024)	✓	✗	✗	✗
PowerSoftmax (2026)	✗	✓	✓	✗
Polynomial attention (2024)	✓	✗	✓	✗
Scaled Signed Averaging (2026)	✗	✗	✓	✓
L1p (ours)	✓	✓	✓	✓

As Table 27 shows, L1p combines signed scores, magnitude invariance, and a learned exponent above one. Each of the most closely related normalizing functions lacks at least one of these properties. Normalized Attention Pooling and ℓ_1 normalization fix the exponent at one. PowerSoftmax and the polynomial attention of Saratchandran et al. (2024) use a fixed exponent above one. PowerSoftmax also keeps its scores non-negative, and the polynomial attention divides by a constant rather than by a sum that scales with the products. Scaled Signed Averaging learns its exponent, but its scores are non-negative and its magnitude invariance holds only asymptotically. It is nonetheless the closest to L1p in motivation, because it is the only other normalizing function that combines a learnable exponent with the sign of the query–key product.

Two findings in this literature relate to our design. The first concerns the placement of signed layers. Both L1p and Cog Attention leave the first layer unsigned but differ in the remaining layers. Cog Attention keeps softmax in the first and last one to three layers, depending on the size of the model, and uses signed scores in every layer in between. L1p instead keeps the first half of its layers unsigned and uses the signed form in every layer of the second half, including the last. As discussed above, Lv et al. (2024) attributed the slower convergence with signed scores in every layer to signs being uninformative early in training. Two designs do not suffice to show that a signed first layer hinders training for signed normalizing functions in general. Appendix E.4 tests the placement of the signed form of L1p directly.

The second finding concerns the difficulty that Saratchandran et al. (2024) reported in training models with exponents above 3. With a constant divisor, as in their polynomial attention, a larger exponent maps a whole row of small products closer to zero. A divisor equal to the sum of the same powers, as in L1p, cancels this shrinkage. For $p \geq 2$, the bound on the Jacobian of L1p nonetheless grows in proportion to the exponent and diverges as the whole row of products approaches zero, the case that Qin et al. (2022) identified as a source of instability.

G.8 Evaluation methodology

A normalizing function that replaces softmax is also expected to maintain the language modeling performance of softmax. Linear attention underperformed softmax on language modeling, sometimes substantially (Qin et al., 2022). In a related result, Deng et al. (2025) showed that softmax can distinguish inputs that sum-normalized query–key products cannot. To close the gap between linear attention and softmax in language modeling, Zhang et al. (2024) designed feature maps that make linear attention behave more like softmax. We accordingly evaluate L1p on language modeling as well as on reasoning depth.

The acquisition of a compositional skill is often abrupt, with models remaining near chance accuracy for a long plateau before reaching high accuracy within a short window. When this transition occurs long after the model has fit its training data, the phenomenon is known as grokking (Power et al., 2021). Liu et al. (2023) related the transition to weight decay and to the geometry of the loss landscape, and Varma et al. (2023) related it to a competition between memorizing and generalizing circuits. Both works reported that the transition depends on the training seed. Liu et al. (2023) observed this dependence in the time that generalization takes. In the analysis of Varma et al. (2023), the seed affected the efficiency of the competing circuits and hence the size of the dataset at which the transition occurs. If a skill is acquired abruptly and the transition depends on the seed, a single run shows only whether one seed crossed the transition within the training budget. The quantity of interest is instead the fraction of seeds that cross it.

This dependence on the seed motivates a departure from the evaluation protocol of Allen-Zhu (2025). On the multi-hop task from which ours is adapted, that work selected the best of four learning rates for each configuration under a single data seed. Only one architecture was trained with 8 seeds, on real data, to measure how much its results varied across seeds. Figure 1 of that work concerns models with about 1 billion parameters trained on 100 billion tokens. In that figure, results varied by at least 2 to 4% across random seeds and by 1 to 2% across architectures. The variation across seeds was thus at least as large as the variation across architectures. Our main comparisons instead use 32 seeds per normalizing function on the reasoning task and up to 24 on language modeling, and we compare the resulting distributions.

The broader literature on evaluation supports comparing distributions over many seeds. Dodge et al. (2019) advocated reporting distributions rather than single best results, and Card et al. (2020) showed that many published comparisons were underpowered for the effects they reported. Bui et al. (2025) measured how much the results of fine-tuning vary when only the training seed changes. Hochlehnert et al. (2025) found that reported gains on reasoning benchmarks were sensitive to evaluation details and to variation at decoding time. Appendix A lists the seeds of every experiment and describes the statistical procedure for each comparison.

Three of the studies discussed above shape our evaluation. The multi-hop reasoning task is adapted from Allen-Zhu (2025), the reporting of accuracy for each hop count follows Curth et al. (2026), and the comparison of several attention mechanisms follows Duvvuri et al. (2026).

To the best of our knowledge, no prior normalizing function combines signed scores, magnitude invariance, and a learned exponent above one. This paper compares L1p with other normalizing functions while holding the rest of the attention layer fixed. The comparison covers reasoning depth and language modeling, with multiple seeds for each normalizing function.