

## A Additional Aspect Ratio Calculations

Based on the Samba+YOCO architecture, we can derive the iso-parametric equation through calculating the number of non-embedding parameters as follows.

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4 + 2dw \cdot w_{\text{attn}}/2, \quad N_{\text{mamba}}(d) = 6dw^2/4,$$

$$N(d) = N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) = 208\alpha d^3 + 13.5\alpha^2 d^3 = 237568d^3.$$

Solving for  $\alpha$ , we get  $\alpha_2 \approx 126$ . For the SambaY+DA architecture, the aspect ratio is not changed because the number of extra parameters introduced by DA is negligible. For MambaY, we have

$$N_{\text{attn}}(d) = 2dw \cdot w_{\text{attn}}/4, \quad N_{\text{mamba}}(d) = 6dw^2/2, \quad N_{\text{gmu}}(d) = 4dw^2/4,$$

$$N(d) = N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) = 64\alpha d^3 + 16\alpha^2 d^3 = 237568d^3.$$

Solving for  $\alpha$ , we get  $\alpha_3 \approx 120$ . For SambaY-MLP, we have

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4 + 2dw \cdot w_{\text{attn}}/4, \quad N_{\text{mamba}}(d) = 6dw^2/4, \quad N_{\text{gmu}}(d) = 8dw^2/4,$$

$$N(d) = N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) = 144\alpha d^3 + 15.5\alpha^2 d^3 = 237568d^3.$$

Solving for  $\alpha$ , we get  $\alpha_4 \approx 120$ . For SambaY-Attn, we have

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4 + 2dw \cdot w_{\text{attn}}/4, \quad N_{\text{mamba}}(d) = 6dw^2/4, \quad N_{\text{gmu}}(d) = 2dw \cdot w_{\text{attn}}/4,$$

$$N(d) = N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) = 208\alpha d^3 + 13.5\alpha^2 d^3 = 237568d^3.$$

Solving for  $\alpha$ , we get  $\alpha_5 \approx 126$ , which is the same as Samba+YOCO. For SambaY-Attn-All, we similarly have

$$N_{\text{attn}}(d) = 2.5dw \cdot w_{\text{attn}}/4, \quad N_{\text{mamba}}(d) = 6dw^2/4, \quad N_{\text{gmu}}(d) = 2dw \cdot w_{\text{attn}}/2,$$

$$N(d) = N_{\text{attn}}(d) + N_{\text{mamba}}(d) + N_{\text{mlp}}(d) + N_{\text{gmu}}(d) = 208\alpha d^3 + 13.5\alpha^2 d^3 = 237568d^3.$$

Solving for  $\alpha$ , we get  $\alpha_6 \approx 126$ .

## B Ablation Study on Hyper-parameter Scaling Laws

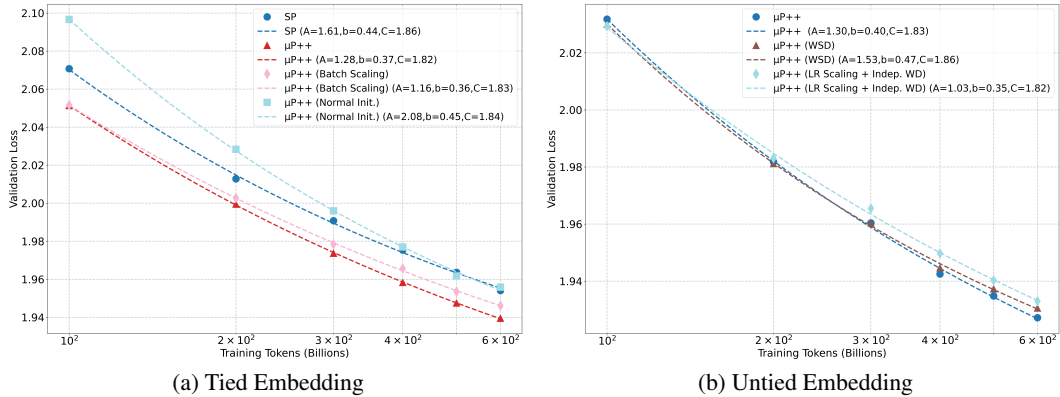


Figure 5: Validation Loss v.s. Training Tokens on the SlimPajama dataset for Transformer++ trained with tied (left) or untied (right) embedding layers.

We conduct a comprehensive ablation study of our  $\mu\text{P}++$  scaling laws to validate their scaling behavior. All experiments are performed using Transformer++ trained with a 4K sequence length on the SlimPajama dataset. To ensure that the linear learning rate scheduler fully decays to zero, we train six models at different training token budgets:  $\{100\text{B}, 200\text{B}, \dots, 600\text{B}\}$  for each of the scaling laws. We examine the scaling performance under both tied and untied embedding setups. For the untied setting, we follow RWKV [PAA<sup>+</sup>23] by applying normal initialization with zero mean and standard deviation of  $10^{-4}$ . The unembedding layer is initialized to zero, following the zero-out trick

proposed in  $\mu\text{P}$  [YHB<sup>+</sup>22]. We first explore batch size scaling with respect to training token size, following [SWW<sup>+</sup>24, LZH<sup>+</sup>25], *i.e.*

$$B = B_0 \sqrt{\frac{T}{T_0}}.$$

As in Figure 5a,  $\mu\text{P}++$  (Batch Scaling) shows both worse learning efficiency and irreducible loss than  $\mu\text{P}++$ . Generally, we think the batch size mainly affects parallelism and the computation efficiency as long as the batch size is not too large, and we think the worse scaling behavior can be because (1) when scaling up, the batch size can surpass the critical batch size [MKAT18], which leads to worse model performance, (2) other optimizer hyper-parameters are not adjusted accordingly with batch size, as in [MLPA22] and leave it for future works to study the large batch size training with  $\mu\text{P}++$ . We also try using Normal Initialization with 0.02 standard deviation for the weight matrices, and scale the variance with respect to  $1/d$ . However,  $\mu\text{P}++$  (Normal Init.) shows worse scaling than  $\mu\text{P}++$ , indicating that it is better to adjust the initialization scaling based on each matrix' dimension as adopted by LeCun initialization, rather than a global factor related to model width. We then try to integrate the empirical scaling law of the learning rate  $\eta$  scaling with respect to training tokens  $T$  [BBC<sup>+</sup>25],

$$\eta = \eta_0 \sqrt{\frac{Bd_0}{B_0d}} \left( \frac{T_0}{T} \right)^{\frac{1}{3}},$$

583 and adjust weight decay to maintain the same regularization effect across different training tokens  
584 with the setup of Independent Weight Decay [WLX<sup>+</sup>24]m.

$$\lambda = \lambda_0 \frac{\eta_0}{\eta},$$

585 where  $\lambda$  is the weight decay in AdamW [LH18] and  $\lambda_0 = 0.1$ . We denote this scaling law as  $\mu\text{P}++$   
586 (LR scaling + Indep. WD). As in Figure 5b, while the irreducible loss is comparable, we observe a  
587 worse learning efficiency with smaller  $b$  compared to  $\mu\text{P}++$ . We think that future work is needed to  
588 have an empirical study of the learning rate scaling with respect to dataset size under  $\mu\text{P}++$ , instead  
589 of transferring the empirical law directly to our theoretical laws. We also explore using the WSD  
590 [HTH<sup>+</sup>24] learning rate scheduler for  $\mu\text{P}++$ , where we set the final decay period to be 2/7 of the  
591 total period following [DA25]. Unfortunately, it depicts worse scaling behavior than  $\mu\text{P}++$  with  
592 linear learning rate schedule, as shown in Figure 5b.

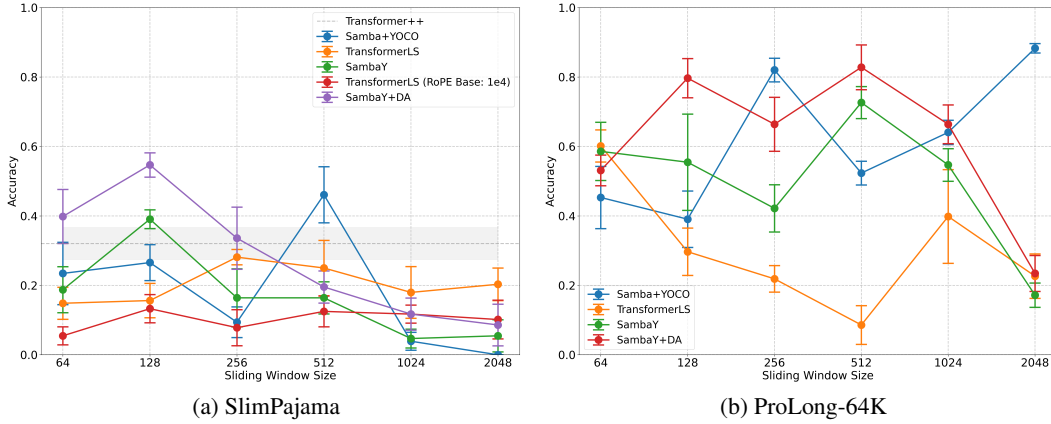


Figure 6: Accuracy (with error bars) v.s. Sliding Window Size on Phonebook with 32K evaluation length using 40B training tokens from SlimPajama (left) or ProLong-64K (right). As an ablation, variable-length training is not applied for both setting.

## 593 C Additional Details on Scaling Comparisons

594 All models are trained with 4K sequence length for drawing the scaling curves. For Standard  
595 Parameterization (SP), we don't apply any  $\mu\text{P}++$  scaling laws, and since LeCun initialization al-  
596 ready scales its initialization variance with respect to  $1/d_{in}$  as proposed in  $\mu\text{P}$ , where  $d_{in}$  is the

fan-in dimension of the weight matrix, we use normal initialization with a standard deviation of 0.02 for weight matrices to rule out this confounding effect. We divide the initialization standard deviation of the output projection of each layer by  $\sqrt{2d}$ , following [RWC<sup>+</sup>19, GD23, RLL<sup>+</sup>25]. The detailed architecture and optimization setups for each of the scales are shown in Table 6. Following [GD23, YWZ<sup>+</sup>24, RLL<sup>+</sup>25, YKH25], our downstream evaluations are conducted on the following benchmarks: Wikitext [MXBS16], LAMBADA (LMB) [PKL<sup>+</sup>16], Arc-Easy/Challenge (ARC-e/ARC-c)[CCE<sup>+</sup>18], HellaSwag (Hella.)[ZHB<sup>+</sup>19], WinoGrande (Wino.) [SBBC21] and PIQA [BZB<sup>+</sup>20], where we measure character normalized accuracy (acc\_n) for Arc-Challenge and HellaSwag.

## D Additional Long-context Retrieval Experiments

Table 6: Architecture Details for Model Configurations. TransformerLS shares the same architecture as Transformer++, employing SWA in all attention layers except every fourth, which uses full attention. MLP Size refers to the intermediate dimension of the MLP, *i.e.*, the input dimension of the output projection. Phi4-mini and Phi4-mini-Flash are trained with a batch size of 8M tokens. Both models use linear learning rate schedule with 3,000 steps warm-up. The product of head dimension and number of query heads is not necessarily tied to the model width. Differential Attention-enhanced variants use the same model configurations as their respective baselines. All models use tied embeddings. The 3.8B-parameter SambaY and Samba+YOCO are initialized with random weights for benchmarking in vLLM.

Architecture	Depth $d$	Model Width	Query Heads	KV Heads	Head Dim	MLP Size	Non-Embed Params (M)	Params (M)	Learning Rate	Training Tokens (B)
Transformer++	8	1024	8	2	128	4096	121.6	154.4	5.66e-04	12.5
	12	1536	12	3	128	6144	410.5	459.7	4.62e-04	42.2
	16	2048	16	4	128	8192	973.1	1038.6	4.00e-04	100.0
	20	2560	20	5	128	10240	1900.5	1982.5	3.58e-04	195.3
	24	3072	24	6	128	12288	3284.1	3382.4	3.27e-04	337.5
SambaY	8	992	8	2	128	3968	123.3	155.0	5.66e-04	12.7
	12	1488	12	3	128	5952	416.1	463.7	4.62e-04	42.8
	16	1984	16	4	128	7936	986.3	1049.8	4.00e-04	101.4
	20	2480	20	5	128	9920	1926.5	2005.8	3.58e-04	198.0
	24	2976	24	6	128	11904	3328.9	3424.2	3.27e-04	342.1
Samba+YOCO	8	1008	8	2	128	4032	123.2	155.4	5.66e-04	12.7
	12	1512	12	3	128	6048	415.6	464.0	4.62e-04	42.7
	16	2016	16	4	128	8064	985.2	1049.7	4.00e-04	101.2
	20	2520	20	5	128	10080	1924.3	2004.9	3.58e-04	197.8
	24	3024	24	6	128	12096	3325.1	3421.9	3.27e-04	341.7
MambaY	16	1920	16	4	128	7680	975.2	1036.6	4.00e-04	40.0
MambaY-2	16	1920	16	4	128	7680	975.2	1036.6	4.00e-04	40.0
SambaY-2	16	1984	16	4	128	7936	986.3	1049.8	4.00e-04	40.0
SambaY-A	16	2016	16	4	128	8064	985.2	1049.7	4.00e-04	40.0
SambaY-AA	16	2016	16	4	128	8064	985.2	1049.7	4.00e-04	40.0
SambaY-MLP	16	1920	16	4	128	7680	985.0	1046.4	4.00e-04	40.0
Phi4-mini	32	3072	24	8	128	8192	3221.2	3835.8	5.00e-04	5000
Pih4-mini-Flash	32	2560	40	20	64	10240	3329.2	3841.4	5.00e-04	5000
SambaY	32	2560	40	20	64	10240	3329.2	3841.4	-	-
Samba+YOCO	32	2560	40	20	64	10240	3224.4	3736.5	-	-

Figure 6 illustrates how different model architectures perform on the Phonebook long-context task as the sliding window size increases, using either SlimPajama or ProLong-64K for pre-training with 32K sequence length, importantly without variable-length training. On SlimPajama, overall accuracy is modest, with SambaY+DA showing some initial promise at smaller window sizes (peaking at 128) before declining, while Samba+YOCO performs best at a moderate window size of 512. Transformer-based models generally struggle to achieve competitive accuracy across window sizes. Switching to ProLong-64K pre-training reveals a significant improvement. Performance is generally improved, most strikingly for Samba+YOCO, which demonstrates a strong positive correlation between accuracy and sliding window size, achieving the highest accuracy at the largest window (2048). SambaY+DA

also performs well, peaking at a 512 window size, but doesn’t scale as effectively as Samba+YOCO with larger windows on this dataset. This suggests that ProLong-64K, even without variable-length training, better equips models like Samba+YOCO to leverage larger context windows. Given that variable-length training on ProLong-64K generally yields even better results as in Figure 3, these fixed-length training results serve as an important ablation. They highlight that while ProLong-64K inherently benefits long-context performance, the full potential, especially for models sensitive to context length, is likely further unlocked by training methodologies that explicitly account for varying sequence lengths of each data sample. The differing optimal sliding window sizes and performance trajectories underscore that both the pre-training dataset and the model architecture significantly influence how effectively the increased sliding window size can be utilized.

## E More Details on Architecture and Large-scale Pre-training

We provide a comprehensive summarization of our architectures and large-scale pre-training setting in Table 6. In our architectures, Differential Attention uses a depth-dependent initialization factor,  $\lambda_{\text{init}} = 0.8 - 0.6 \exp(-0.3 \times l)$ , where  $l$  is the depth index. For each attention head, it employs two sets of learnable parameters,  $(\lambda_{q1}, \lambda_{k1})$  and  $(\lambda_{q2}, \lambda_{k2})$ , each of dimension equal to the head dimension and initialized with a normal distribution of zero mean and 0.1 standard deviation. RMSNorm [ZS19] with learnable element-wise affine parameters is adopted for attention output normalization.

## F Additional Details on Reasoning Results

Following Phi4-mini-Reasoning [XPA<sup>+</sup>25], the evaluation is conducted with a temperature setting of 0.6, a top-p value of 0.95, and a maximum sequence length of 32,768 tokens. Each task involves 3 runs, and the average performance across these trials is reported.

We conduct case studies for Phi4-mini-Flash-Reasoning in Example 1 and Example 2 for both mathematical reasoning and daily-life reasoning. Notably, our model is only trained on distillation data with math problems. We can see our model exhibits advanced reasoning techniques such as double checking, reflection, and backtracking in both cases. Surprisingly, in Example 2, our model will even guess the user intention (*e.g.* “But maybe the user intended “strawberry” with two ‘r’s, but wrote “strawberrry” by mistake.”) to finally provide the correct answer.

### Example 1: Mathematical Reasoning (Part 1)

**Query:** solve  $x^2+3x+4=1$

*Model's reasoning process:*

#### Think

Okay, so I need to solve the equation  $x$  squared plus three  $x$  plus four equals one. Let me write that down first to make sure I have it right:  $x^2 + 3x + 4 = 1$ . Hmm, alright. The first thing I remember is that to solve a quadratic equation, it's usually best to get everything on one side so that the other side is zero. That way, I can factor it or use the quadratic formula if needed. Let me try subtracting 1 from both sides to move that 1 over. So, subtracting 1 from both sides gives me  $x^2 + 3x + 4 - 1 = 0$ . Simplifying that, 4 minus 1 is 3, so the equation becomes  $x^2 + 3x + 3 = 0$ .

Now, I need to check if this quadratic can be factored easily. Let me see. The standard form is  $ax^2 + bx + c$ , so here  $a$  is 1,  $b$  is 3, and  $c$  is 3. To factor it, I need two numbers that multiply to 3 (since  $a$  is 1, it's just  $c$ ) and add up to 3. Let me think... The factors of 3 are 1 and 3. But  $1 + 3$  is 4, which is more than 3. There's also negative numbers, but since both the product and the sum are positive, both numbers need to be positive. But 1 and 3 add to 4, which isn't 3. So factoring doesn't seem straightforward here. Maybe I need to use the quadratic formula instead. The quadratic formula is  $x$  equals negative  $b$  plus or minus the square root of  $b$  squared minus four  $a$   $c$  all over  $2a$ . Let me plug in the values. Here,  $a$  is 1,  $b$  is 3, and  $c$  is 3. So substituting those in, we get  $x = [-3 \pm \sqrt{(3^2 - 4 \cdot 1 \cdot 3)}] / (2 \cdot 1)$ . Let me compute the discriminant first, which is the part under the square root: 3 squared is 9, and 4 times 1 times 3 is 12. So 9 minus 12 is -3. Wait, the discriminant is negative. That means there are no real solutions, only complex ones. But the problem didn't specify whether they want real solutions or if complex solutions are acceptable. Since it's a math problem, unless specified, sometimes they expect complex solutions. Let me proceed. So the square root of -3 can be written as  $i\sqrt{3}$ , where  $i$  is the imaginary unit. Therefore, the solutions would be  $x = [-3 \pm i\sqrt{3}]/2$ .

Let me double-check my steps to make sure I didn't make a mistake. Starting with the original equation:  $x^2 + 3x + 4 = 1$ . Subtract 1:  $x^2 + 3x + 3 = 0$ . Check. Then discriminant:  $b^2 - 4ac = 9 - 12 = -3$ . Correct. So the roots are complex:  $(-3 \pm i\sqrt{3})/2$ .

Alternatively, maybe I can complete the square instead. Let me try that method to verify. Starting with  $x^2 + 3x + 3 = 0$ . Move the constant term to the other side:  $x^2 + 3x = -3$ . To complete the square, take half of the coefficient of  $x$ , which is  $3/2$ , square it to get  $(9/4)$ . Add that to both sides:  $x^2 + 3x + 9/4 = -3 + 9/4$ . The left side becomes  $(x + 3/2)^2$ . The right side:  $-3$  is  $-12/4$ , so  $-12/4 + 9/4$  is  $-3/4$ . So  $(x + 3/2)^2 = -3/4$ . Taking the square root of both sides gives  $x + 3/2 = \pm\sqrt{(-3/4)} = \pm i\sqrt{3}/2$ . Subtract  $3/2$  from both sides:  $x = -3/2 \pm i\sqrt{3}/2$ . Which is the same as  $(-3 \pm i\sqrt{3})/2$ . So that matches the quadratic formula result.

Therefore, the solutions are complex numbers:  $(-3 + i\sqrt{3})/2$  and  $(-3 - i\sqrt{3})/2$ . Since the original equation is a quadratic, there are two solutions, and they are complex conjugates of each other. I should also check if there was a mistake in the problem statement. The original equation was  $x^2 + 3x + 4 = 1$ . If I plug the solutions back into the equation, they should satisfy it. Let me test one of them. Let's take  $x = [-3 + i\sqrt{3}]/2$ . Plugging into  $x^2 + 3x + 4$ . First, compute  $x^2$ :

$x = (-3 + i\sqrt{3})/2$ . So  $x$  squared is  $[(-3 + i\sqrt{3})/2]^2$ . Let's compute that. First, square the numerator:  $(-3 + i\sqrt{3})^2$ . That's  $(-3)^2 + 2 \cdot (-3) \cdot (i\sqrt{3}) + (i\sqrt{3})^2$ . Which is  $9 - 6i\sqrt{3} + (i^2 \cdot (\sqrt{3})^2)$ . Since  $i^2$  is -1, that term becomes  $-1 \cdot 3 = -3$ . So  $9 - 6i\sqrt{3} - 3 = 6 - 6i\sqrt{3}$ . Then divide by  $2^2=4$ . So  $x^2 = (6 - 6i\sqrt{3})/4 = (3 - 3i\sqrt{3})/2$ .

Now compute  $3x$ :  $3 \cdot [(-3 + i\sqrt{3})/2] = (-9 + 3i\sqrt{3})/2$ .

Adding  $x^2 + 3x + 4$ :  $(3 - 3i\sqrt{3})/2 + (-9 + 3i\sqrt{3})/2 + 4$ . Combine the fractions:  $[3 - 3i\sqrt{3} - 9 + 3i\sqrt{3}]/2 + 4$ . The imaginary parts cancel out:  $(3 - 9)/2 + 4 = (-6)/2 + 4 = -3 + 4 = 1$ . Which matches the right-hand side of the equation. So that works. Similarly, the other root would also satisfy the equation.

Therefore, the solutions are correct. So the answer is  $x$  equals  $(-3$  plus or minus  $i\sqrt{3})$  over 2.

I think that's all. The key steps were moving 1 to the left to set the equation to zero, recognizing that the quadratic doesn't factor nicely, computing the discriminant, finding it's negative, and then proceeding with the quadratic formula or completing the square to find the complex solutions. Then verifying by plugging back in. So the final answer should be the two complex numbers.

#### Model's Response:

*To be continued...*

### Example 1: Mathematical Reasoning (Part 2)

#### Model's Response (continued):

To solve the equation  $x^2 + 3x + 4 = 1$ :

1. \*\*Rearrange the equation\*\* to standard quadratic form:

$$x^2 + 3x + 4 - 1 = 0 \implies x^2 + 3x + 3 = 0$$

2. \*\*Identify coefficients\*\*:  $a = 1, b = 3, c = 3$ .

3. \*\*Compute the discriminant\*\*:

$$\Delta = b^2 - 4ac = 3^2 - 4(1)(3) = 9 - 12 = -3$$

Since the discriminant is negative, the solutions are complex.

4. \*\*Apply the quadratic formula\*\*:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a} = \frac{-3 \pm \sqrt{-3}}{2} = \frac{-3 \pm i\sqrt{3}}{2}$$

5. \*\*Verify by completing the square\*\*:

$$x^2 + 3x = -3 \implies \left(x + \frac{3}{2}\right)^2 = -\frac{3}{4}$$

Taking square roots:

$$x + \frac{3}{2} = \pm \frac{i\sqrt{3}}{2} \implies x = -\frac{3}{2} \pm \frac{i\sqrt{3}}{2}$$

This confirms the solutions.

**\*\*Solutions\*\*:**

$$x = \frac{-3 + i\sqrt{3}}{2} \quad \text{and} \quad x = \frac{-3 - i\sqrt{3}}{2}$$

$$x = \frac{-3 \pm i\sqrt{3}}{2}$$

## Example 2: Letter Counting in Misspelled Words

**Query:** how many r's in "strawberry"?

*Model's reasoning process:*

### Think

Okay, let's see. The question is asking how many 'r's are in the word "strawberry". Alright, first I need to look at the word carefully. Let me write it out: S-T-R-A-W-B-E-R-R-Y. Wait, the original word is "strawberry". So, the spelling has three 'r's in a row, right? Let me count them one by one.

Starting from the beginning: S (1), T (2), R (3), A (4), W (5), B (6), E (7), R (8), R (9), Y (10). Hmm, so between the B and Y, there are three 'r's. Let me check again. The word is spelled as S-T-R-A-W-B-E-R-R-Y. So yes, after the B and E, there are three 'r's. So that's three 'r's in a row. Wait, but when I count the letters, let me make sure I'm not missing any. Let's break it down:

S T R A W B E R R R Y Wait, so actually, there are four 'r's? Wait, no. Wait, the original word is "strawberry". Let me count the letters in "strawberry". Let's spell it out: S-T-R-A-W-B-E-R-R-R-Y. Let's count each letter:

S T R A W B E R R R Y So that's 11 letters total. Now, the 'r's are at positions 3, 8, 9, 10. So that's four 'r's? Wait, but the word is written as "strawberry". So maybe the user intended three 'r's? Wait, but when I look at the spelling, "strawberry" normally has two 'r's. But here it's "strawberry" with three 'r's. Wait, maybe the correct spelling is "strawberry" with two 'r's. But the user wrote "strawberry" with three 'r's. So in that case, the answer would be three 'r's. Wait, but when I count the letters in "strawberry", let's see: S-T-R-A-W-B-E-R-R-R-Y. So that's three 'r's in a row. So total 'r's would be three. Wait, but when I count the letters, positions 3, 8, 9, 10. Wait, that's four 'r's. Wait, maybe I'm miscounting. Let me write the word out:

S (1), T (2), R (3), A (4), W (5), B (6), E (7), R (8), R (9), R (10), Y (11). So positions 3,8,9,10 are 'r's. That's four 'r's. Wait, but the word is "strawberry". Let me check the spelling again. The user wrote "strawberry". So between B and Y, there are three 'r's. So the word is spelled as S-T-R-A-W-B-E-R-R-R-Y. So that's three 'r's after the B and E. So total 'r's: the first 'r' is at position 3, then three more at positions 8,9,10. Wait, that's four 'r's in total. Wait, but maybe the user made a typo. Let me confirm. The word is "strawberry". So "straw" + "berry". So "straw" is S-T-R-A-W, then "berry" would be B-E-R-R-R-Y. So in "berry", there are three 'r's. So total in the whole word: the first 'r' in "straw", then three 'r's in "berry". So total of four 'r's. Wait, but the original word is "strawberry". Let me check again. S-T-R-A-W-B-E-R-R-R-Y. So yes, four 'r's. But maybe the user intended "strawberry" with two 'r's, but wrote "strawberry" by mistake. But the question is as given, so I should answer based on the exact spelling provided. So in "strawberry", how many 'r's? Let's count each occurrence:

Looking at each letter:

s, t, r, a, w, b, e, r, r, r, y.

So the 'r's are the third letter, then the eighth, ninth, and tenth letters. That's four 'r's. Wait, but maybe I'm miscounting. Let me list them:

S T R A W B E R R R Y Yes, so positions 3,8,9,10. That's four 'r's. So the answer should be 4. But wait, maybe the user intended "strawberry" which has two 'r's. But the question specifically says "strawberry", so I have to go with that. Therefore, the number of 'r's is 4. Wait, but let me check once more. Let's spell it out: S-T-R-A-W-B-E-R-R-R-Y. So after B and E, there are three 'r's. So that's three 'r's in the middle, plus the initial 'r' in "straw". So total of four. Yes. So the answer is 4.

### Model's Response:

The word "strawberry" is spelled as S-T-R-A-W-B-E-R-R-R-Y. Breaking it down:

Letters: S, T, R, A, W, B, E, R, R, R, Y 'r's: The third letter is 'r', followed by three more 'r's later in the word. Total number of 'r's: 4.

4



## G Related Works

**KV Cache Sharing.** Efficient inference in transformer-based models has been significantly advanced through techniques that reduce memory consumption, particularly concerning key-value (KV) caching. Traditional approaches like Multi-Query Attention (MQA) [Sha19] and Grouped-Query Attention (GQA) [ALTdJ+23] have enabled multiple query heads to share a single key/value head within the same layer, effectively reducing the number of distinct key/value heads with minimal impact on accuracy. Apart from YOCO [SDZ+24], Cross-Layer Attention (CLA) [BMN+24] extends KV sharing across adjacent layers, achieving up to two times reduction in KV cache size while maintaining performance. Our work focuses on representation sharing across SSM layers, directly sharing the output from the SSM kernel to avoid materializing recurrent states, thereby preserving the parallel training efficiency of SSM layers.

**Efficient Long Generation.** Efficient long-sequence generation in transformer models has been a focus of recent research on LLM efficiency, primarily due to the substantial memory demands associated with key-value (KV) caching during inference with long CoTs [KKL20, WWZ+24, YCQ+21, DYL+24]. To address these challenges, several techniques have been proposed to optimize memory usage without compromising model performance. One notable approach is the Layer-Condensed KV Cache (LCKV) [WT24], which computes and caches KV pairs for only a subset of layers, significantly reducing memory consumption and improving inference throughput. Another advancement is InfiniGen [LLSS24], a dynamic KV cache management framework that selectively prefetches essential KV cache entries, thereby mitigating fetch overhead from host memory in offloading-based LLM serving systems. These methods collectively contribute to more efficient long-sequence generation by optimizing KV cache usage, and are orthogonal to our works, as we can also apply these techniques to improve the memory I/O efficiency of our full attention layer.

**Scaling Laws.** Understanding how model performance scales with size and data is crucial for efficient large-scale training. Empirical studies have shown that transformer models exhibit predictable scaling behaviors, where performance improves with increased model parameters and training data [HNA+17, KMH+20, BDK+24, ANZ22, HBM+22]. Numerous works have also investigated scaling laws for hyper-parameters, based on either empirical studies [BBC+25, WLX+24] or theoretical analyses [MLPA22, YHB+22, YYZH23, WA24]. In this work, we focus on theoretical hyper-parameter scaling laws since they are not over-tuned for the Transformer architectures, so they could provide fairer comparisons for the emerging architectures.

## H Limitation

We validate our model’s reasoning capability using distillation-based Supervised Fine-Tuning (SFT), but Reinforcement Learning (RL) remains under-explored in the context of hybrid architectures. Due to resource constraints, we do not perform an exhaustive hyperparameter search for each architecture. Instead, we adopt a generic optimization setup based on Transformer++ for learning rate, weight decay, warm-up schedule, batch size, AdamW betas, epsilon, and other parameters. It is likely that aggressive tuning of these optimization settings could yield improved results. We leave a more comprehensive study of the interplay between optimization setups and architecture designs for future work. Lastly, our architecture still includes a full-attention layer, which leads to linear computational complexity during decoding. This underscores future research direction on designing models for extremely long sequence generation that can maintain constant decoding complexity while effectively leverage long-context memory.