Pretraining Language Models to Ponder in Continuous Space

Boyi Zeng[1], **Shixiang Song**[1,2,3], **Siyuan Huang**[1], **Yixuan Wang**[1,3], **He Li**[1], **Ziwei He**[3],**Xinbing Wang**[4], **Zhiyu Li**[2], **Zhouhan Lin**[1,2,3*]
[1]LUMIA Lab, Shanghai Jiao Tong University,
[2]Institute for Advanced Algorithms Research, Shanghai,
[3]Shanghai Innovation Institute, [4]Shanghai Jiao Tong University
`boyizeng@sjtu.edu.cn` *`lin.zhouhan@gmail.com`

Abstract

Humans ponder before articulating complex sentence elements, enabling deeper cognitive processing through focused effort. In this work, we introduce this pondering process into language models by repeatedly invoking the forward process within a single token generation step. During pondering, instead of generating an actual token sampled from the prediction distribution, the model ponders by yielding a weighted sum of all token embeddings according to the predicted token distribution. The generated embedding is then fed back as input for another forward pass. We show that the model can learn to ponder in this way through self-supervised learning, without any human annotations. Our method is straightforward and can be seamlessly integrated with various existing language models. Experiments across three widely used open-source architectures—GPT-2, Pythia, and LLaMA—and extensive downstream task evaluations demonstrate the effectiveness and generality of our method. For language modeling tasks, pondering language models achieve performance comparable to vanilla models with twice the number of parameters. On 9 downstream benchmarks, our pondering-enhanced Pythia models significantly outperform the official Pythia models. Notably, pondering-enhanced Pythia-1B is comparable to TinyLlama-1.1B, which is trained on 10 times more data.[1]

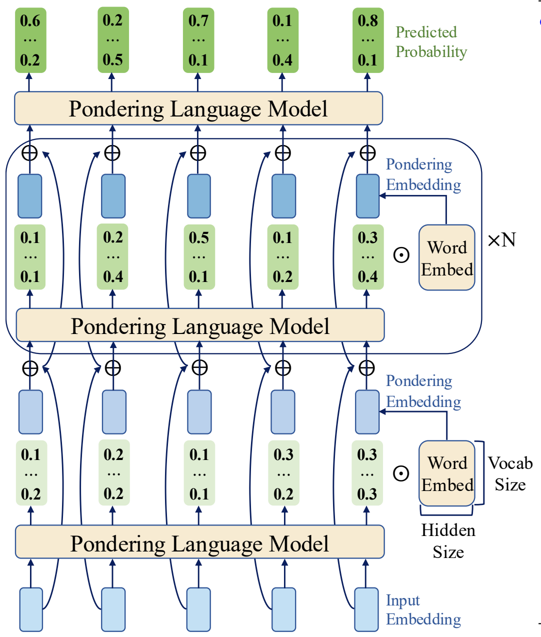
1 Introduction

In the pursuit of improving model performance, scaling up model parameters and data sizes has long been the most widely adopted and accepted approach (Kaplan et al., 2020; Brown et al., 2020; Liu et al., 2024). However, this approach faces several bottlenecks, including the exhaustion of high-quality data (Villalobos et al., 2022; Muennighoff et al., 2023), the observed saturation in scaling laws (Hackenburg et al., 2025; Hoffmann et al., 2022a) and substantial communication overhead in distributed pre-training that grows super-linearly with model size (Narayanan et al., 2021; Pati et al., 2023; Li et al., 2024).

On the other hand, if we look at humans, the growth of human capabilities does not stem from simply increasing the number of neurons in the brain. Instead, when faced with complex problems, humans often enhance their problem-solving abilities by repeatedly pondering, engaging in deep cognitive processing before articulating their thoughts.

Analogously, in large language models, the most relevant research direction is test-time scaling. In particular, following the advancements in o1 and R1 (Jaech et al., 2024; DeepSeek-AI et al., 2025), generating long chains of thought (CoT) has emerged as the mainstream approach for scaling test-time computation. However, CoT-based methods also exhibit several drawbacks: they often require curated human-annotated datasets (Allen-Zhu & Li, 2023) and carefully designed reinforcement learning

[*]Zhouhan Lin is the corresponding author.
[1]The code is available at `https://github.com/LUMIA-Group/PonderingLM`.



```
class PonderingLanguageModel(nn.Module):
    def __init__(self, lm, v, h, k):
        self.lm = lm # language model
        self.vocab_size = v
        self.hidden_dim = h
        self.pondering_steps = k
        self.embedding = nn.Parameter(torch.
            randn(v, h), requires_grad=True)

    def forward(self, input_tokens):
        input_embedding =
        self.embedding[input_tokens]
        #Iterative pondering
        for t in range(self.pondering_steps):
            predicted_prob = self.lm(
                input_embedding)
            pondering_embedding = torch.
                matmul(predicted_prob, self.
                embedding)
            input_embedding = input_embedding
                 + pondering_embedding
        #Final forward pass
        final_prob = self.lm(input_embedding)
        return final_prob
```

Figure 1: Overview of the Pondering Language Model. Given input token embeddings, the base LM produces a probability distribution over the vocabulary, which is used to compute a continuous "pondering embedding" via a weighted sum of all token embeddings. This embedding is then added residually to the original input embeddings and fed back into the LM. By repeating this process for k steps within a single token prediction, the model iteratively refines its output distributions. The pseudocode on the right illustrates the implementation details.

algorithms (Pang et al., 2025). Moreover, small models rarely benefit from CoT (Li et al., 2023), and the upper bound of performance remains constrained by the base pretrained model (Yue et al., 2025). Additionally, current language models employing CoT are still confined to discrete language spaces with fixed vocabularies, which, according to recent studies (Fedorenko et al., 2024; Hao et al., 2024; Pfau et al., 2024), are primarily optimized for communication rather than for internal computational thinking.

To overcome these challenges and inspired by human pondering, we introduce the Pondering Language Model (Pondering LM), which relies solely on self-supervised learning. Pondering LMs can be naturally learned through pretraining on large-scale general corpora, without the need for human-annotated datasets or reinforcement learning.

During pondering, instead of generating a discrete token sampled from the prediction distribution, the model produces a weighted sum of all token embeddings based on the predicted probabilities. This generated embedding is then fed back into the language model, allowing it to iteratively refine its predictions. As the weighted embedding is continuous, Pondering LMs overcome the expressive limitations of discrete token vocabularies and enable fully differentiable, end-to-end pretraining via gradient descent. Furthermore, by performing more computations per parameter, Pondering LMs achieve higher parameter knowledge density (Allen-Zhu & Li, 2024), potentially reducing communication costs at scale.

Experimentally, by introducing the pondering process during pretraining, our Pondering GPT-2, LLaMA, and Pythia models achieve pretraining perplexity comparable to that of vanilla models with twice as many parameters. Furthermore, Pondering Pythia models significantly outperform the official Pythia models across nine popular downstream tasks, and PonderingPythia-1B is comparable to TinyLlama-1.1B, which is trained on 10 times more data. Notably, increasing the number of pondering steps consistently enhances model performance, underscoring the substantial potential of this approach.

Moreover, our method is orthogonal to traditional scaling strategies, including parameter scaling and inference-time scaling via CoT, and thus can complement existing techniques, potentially introducing a third scaling axis to enhance model performance.

2 Pondering Language Model

In this section, we introduce our proposed Pondering Language Model (Figure 1), which integrates a pondering mechanism into language models via pretraining. Given that pretraining fundamentally constitutes a language modeling task, we briefly review this task before detailing our proposed model.

Language Modeling. Given a sequence of tokens $X = [x_1, x_2, \dots, x_n]$, the primary objective of language modeling is typically to maximize the likelihood of predicting each token based on its preceding tokens. Formally, this is expressed through the joint probability factorization:

$$P(x_1, x_2, \dots, x_n) = \prod_{t=1}^{n} P(x_t \mid x_{<t}) \tag{1}$$

Current language models first map tokens to input embeddings $\mathbf{E}^0 = [\mathbf{e}_1^0, \mathbf{e}_2^0, \dots, \mathbf{e}_n^0]$, where each embedding $\mathbf{e}_i^0 \in \mathbb{R}^d$ is selected from a vocabulary embedding matrix $\mathbf{V} = [\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_{|V|}]$, with vocabulary size $|V|$ and hidden dimension d. The language model then generates output probabilities $\mathbf{P}$ for predicting the next token at each position:

$$\mathbf{P} = \mathrm{LM}(\mathbf{E}^0), \quad \mathbf{P} \in \mathbb{R}^{n \times |V|} \tag{2}$$

The cross-entropy loss is computed directly from these predicted probabilities $\mathbf{P}$ to pretrain the language model.

Pondering Mechanism. In our proposed method, instead of directly using the predicted output probabilities $\mathbf{P}$ to compute the cross-entropy loss, we utilize these probabilities as weights to sum embeddings of all candidate tokens, forming what we call a "pondering embedding". Given the probability distribution $\mathbf{p} \in \mathbb{R}^{|V|}$ at each position, the pondering embedding $\mathbf{t}$ is:

$$\mathbf{t} = \sum_{i=1}^{|V|} p_i \mathbf{e}_i, \quad p_i \in \mathbb{R}, \mathbf{e}_i \in \mathbb{R}^d \tag{3}$$

For computational efficiency, pondering embeddings $\mathbf{t}$ for all positions can be calculated simultaneously via matrix multiplication[2]:

$$\mathbf{T} = \mathbf{P}\mathbf{V}, \quad \mathbf{T} \in \mathbb{R}^{n \times d} \tag{4}$$

Through these pondering embeddings, we effectively map predicted probabilities back into the embedding space, preserving embedding information from all possible candidate tokens. To maintain the information from the original input embeddings, we integrate the pondering embeddings using a residual connection:

$$\mathbf{E}^1 = \mathbf{E}^0 + \mathbf{T} = [\mathbf{e}_1^0 + \mathbf{t}_1, \mathbf{e}_2^0 + \mathbf{t}_2, \dots, \mathbf{e}_n^0 + \mathbf{t}_n] \tag{5}$$

We then feed the updated embeddings $\mathbf{E}^1$ back into the same language model to obtain refined output probabilities:

$$\mathbf{P}^1 = \mathrm{LM}(\mathbf{E}^1), \quad \mathbf{P}^1 \in \mathbb{R}^{n \times |V|} \tag{6}$$

After obtaining $\mathbf{P}^1$, we can iteratively repeat the previous process to achieve multi-step pondering. Specifically, given a predefined number of pondering steps k[3], we iteratively compute new pondering embeddings and integrate them with the original input embeddings using residual connections, feeding the result back into the same language model until k steps are reached:

$$\begin{aligned}
&\mathbf{E}^0 = [\mathbf{e}_1^0, \mathbf{e}_2^0, \dots, \mathbf{e}_n^0], \quad \mathbf{P}^0 = \mathrm{LM}(\mathbf{E}^0), \quad \mathbf{T}^1 = \mathbf{P}^0\mathbf{V} \\
&\mathbf{E}^1 = \mathbf{E}^0 + \mathbf{T}^1 = [\mathbf{e}_1^0 + \mathbf{t}_1^1, \mathbf{e}_2^0 + \mathbf{t}_2^1, \dots, \mathbf{e}_n^0 + \mathbf{t}_n^1], \quad \mathbf{P}^1 = \mathrm{LM}(\mathbf{E}^1), \quad \mathbf{T}^2 = \mathbf{P}^1\mathbf{V} \\
&\quad \cdots \\
&\mathbf{E}^k = \mathbf{E}^0 + \sum_{i=1}^{k} \mathbf{T}^i = [\mathbf{e}_1^0 + \mathbf{t}_1^1 + \cdots + \mathbf{t}_1^k, \dots, \mathbf{e}_n^0 + \mathbf{t}_n^1 + \cdots + \mathbf{t}_n^k], \quad \mathbf{P}^k = \mathrm{LM}(\mathbf{E}^k)
\end{aligned} \tag{7}$$

[2] In practice, we use only the top-K tokens with highest probabilities at each position to compute the pondering embedding, reducing complexity from $\mathcal{O}(N|V|D)$ to $\mathcal{O}(NKD)$. With $K = 100 \ll |V|$, this does not degrade LM performance and makes the matrix multiplication overhead negligible within the overall LM computations.

[3] Unless otherwise specified, we set $k = 3$ for subsequent experiments.

This iterative pondering mechanism progressively refines the model's predictions. Finally, we can use the refined output probabilities $\mathbf{P}^k$ after k pondering steps to compute the cross-entropy loss and optimize the language model to perform k-step pondering.

3 Experiments

Our experiments consist of four parts. First, we validate the scaling curves of pondering models on widely used GPT-2 and LLaMA architectures. Second, we perform large-scale pretraining of PonderingPythia models on the Pile dataset and compare their scaling curves and language modeling capabilities with those of the official Pythia suite (Biderman et al., 2023). Third, we evaluate the downstream task performance of PonderingPythia models, including nine popular general tasks and an instruction-following task, and compare the results with official Pythia, OPT (Zhang et al., 2022), Bloom (Le Scao et al., 2023), GPT-Neo (Black et al., 2021), and TinyLLaMA (Zhang et al., 2024) models. Finally, we investigate the impact of the number of pondering steps on pretraining perplexity.

3.1 Small Scale Validation on GPT-2 and LLaMA

We apply our proposed method to two popular Transformer architectures, GPT-2 and LLaMA, to investigate its general applicability and effectiveness. Specifically, we plot and compare scaling curves between vanilla GPT-2, vanilla LLaMA, and their corresponding pondering versions, referred to as PonderingGPT and PonderingLLaMA, respectively.

Experimental Settings. We train all models from scratch on a subset of the Pile dataset with the same tokenizer. The model sizes range from 405M to 1.4B parameters, with a context length fixed at 2048 tokens. The number of training tokens for each model is chosen to approximately match the scaling laws proposed by Chinchilla (Hoffmann et al., 2022b).

The detailed configurations of the models, including sizes, learning rates, and batch sizes, are specified in Table 1. These hyperparameters primarily follow the GPT-3 specifications (Brown et al., 2020). However, unlike GPT-3, we untie the input and output embedding matrices.

Table 1: Model sizes and hyperparameters for scaling experiments.

params	n_{layers}	d_{model}	n_{heads}	learning rate	batch size (in tokens)	tokens
405M	24	1024	16	3e-4	0.5M	7B
834M	24	1536	24	2.5e-4	0.5M	15B
1.4B	24	2048	32	2e-4	0.5M	26B

Results. Figure 2 (top) illustrates the scaling curves of the validation loss on the Pile subset for vanilla GPT-2, vanilla LLaMA, and their pondering counterparts. Our results show that incorporating pondering significantly improves the performance for both GPT-2 and LLaMA across the entire size range tested (405M to 1.4B parameters). For instance, by fitting scaling curves, we find that PonderingGPT-834M achieves a loss comparable to a vanilla GPT-2 model trained with approximately $2.01\times$ parameters$*$tokens, while PonderingLLaMA-834M matches the loss of vanilla LLaMA trained with roughly $2.26\times$ parameters$*$tokens.

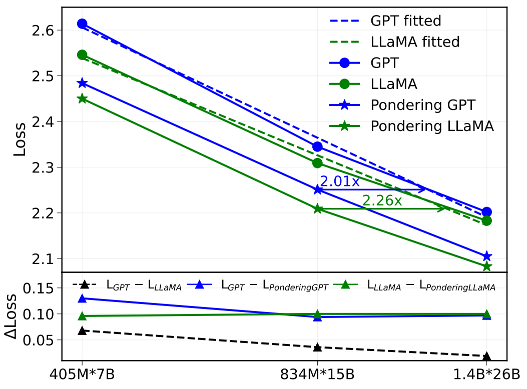


Figure 2: (top) Scaling curves of GPT3 LLaMA and their corresponding pondering models. (bottom) Relative improvements of RoPE + RMSNorm + SwiGLU MLP and Pondering.

Furthermore, Figure 2 (bottom) shows the relative validation loss improvements (denoted as Δ loss) of architectural modifications inherent in LLaMA—namely rotary positional embeddings (RoPE) (Su et al., 2024), RMSNorm (Zhang & Sennrich, 2019), and SwiGLU activation—in comparison to GPT-2, alongside the relative improvements obtained by our pondering method for both

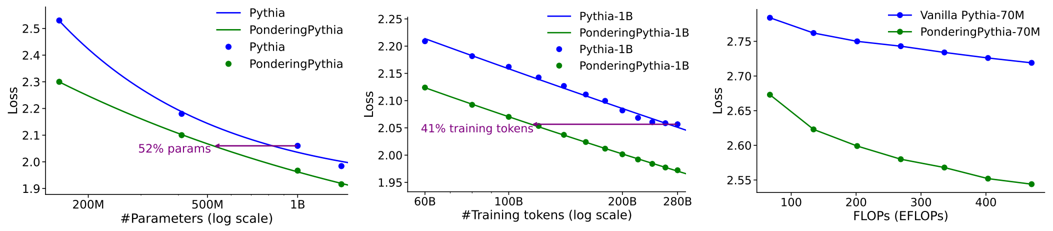


Figure 3: Language modeling loss when scaling parameter count and training tokens. PonderingPythia matches the performance of the official Pythia-1B using only 41% of the training tokens or 52% of the parameters.

Table 2: Language Modeling Perplexity (ppl). Lower is better. The values in parentheses indicate the improvement (↓) compared to the corresponding Pythia model. For simplicity, we refer to PonderingPythia as Ponder.

Model	Pile	Wikitext	Lambada Openai	Lambada Standard
Pythia-14M	35.87	145.06	1787.20	6412.67
Ponder-14M	**24.05**(↓ 11.82)	**87.58**(↓ 57.48)	**410.42**(↓ 1376.78)	**2199.73**(↓ 4212.94)
Pythia-70M	18.36	57.03	141.94	967.72
Ponder-70M	**12.68**(↓ 5.68)	**34.07**(↓ 22.96)	**39.19**(↓ 102.75)	**145.51**(↓ 822.21)
Pythia-160M	12.55	33.43	38.15	186.51
Ponder-160M	**9.87**(↓ 2.68)	**23.69**(↓ 9.74)	**15.96**(↓ 22.19)	**41.24**(↓ 145.27)
Pythia-410M	8.85	20.11	10.86	31.54
Ponder-410M	**8.15**(↓ 0.70)	**17.78**(↓ 2.33)	**8.55**(↓ 2.31)	**17.48**(↓ 14.06)
Pythia-1B	7.85	16.45	7.91	17.41
Ponder-1B	**7.15**(↓ 0.70)	**14.61**(↓ 1.84)	**6.02**(↓ 1.89)	**10.64**(↓ 6.77)
Pythia-1.4B	7.24	14.72	6.08	10.87
Ponder-1.4B	**6.82**(↓ 0.42)	**13.59**(↓ 1.13)	**5.37**(↓ 0.71)	**8.88**(↓ 1.99)

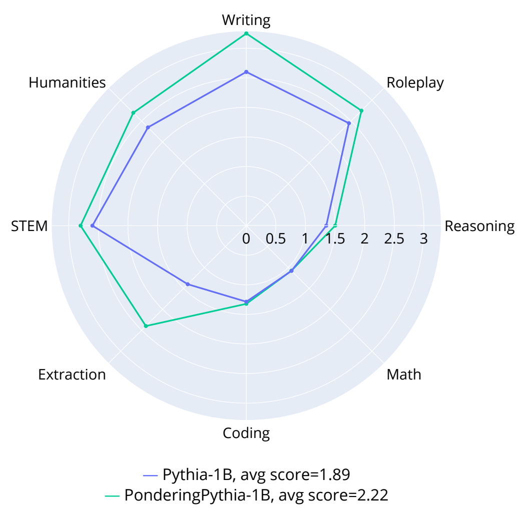
architectures. We observe that while the relative improvement due to RoPE, RMSNorm, and SwiGLU MLP diminishes as model size and data scale increase (reaching approximately 0.02 for the 1.4B parameter model), the relative improvement provided by pondering consistently remains around 0.1 across the entire tested scale.

3.2 Large-Scale Pretraining on Pile

We further validate the effectiveness of our pondering method by conducting large-scale pretraining experiments on the entire Pile dataset (300B tokens) (Gao et al., 2020). We train a new model, named PonderingPythia, from scratch using exactly the same architectural components (parallel attention and MLP layers, rotary embeddings with 1/4 head dimensions), same tokenizer and training hyperparameters (optimizer settings, learning rate schedule, batch size, and context length) as the original Pythia models. We then compare PonderingPythia models' scaling curves and language modeling capabilities with the official Pythia model suite.

3.2.1 Scaling Curves

We plot the PonderingPythia and official Pythia model's scaling curves along parameters size, training tokens and training flops. As depicted in Figure 3 (left), the fitted curves show that a 520M-parameter PonderingPythia model achieves a validation loss comparable to the 1B-parameter official Pythia model, while requiring only 52% of the parameters. In Figure 3 (middle), we evaluated the 1B PonderingPythia and official Pythia models at intervals of 20B training tokens. The fitted curves indicate that PonderingPythia trained with 115B tokens achieves performance comparable to the official Pythia model trained with 280B tokens, consuming only 41% of the



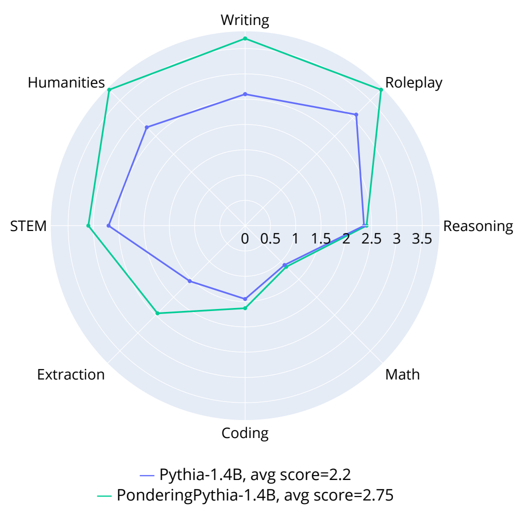


Figure 4: Instruction-following abilities evaluated on MT-Bench. PonderingPythia-1B and 1.4B consistently outperform their corresponding official Pythia models across all subtasks.

training tokens. In Figure 3 (right), we report the language modeling loss of vanilla Pythia-70M[4] and PonderingPythia-70M under the same computational budget during pretraining. It can be observed that PonderingPythia-70M consistently outperforms vanilla Pythia-70M when trained with the same number of FLOPs.

3.2.2 Language Modeling Ability Evaluation

We measure the perplexity on several language modeling datasets to reflect general language modeling capabilities. Specifically, we report perplexity scores on the Pile validation set, Wikitext, and the Lambada dataset (both OpenAI and standard versions), detailed in Table 2. The results demonstrate significant perplexity improvements across all datasets and model sizes. Notably, the perplexity achieved by the PonderingPythi-70M model is comparable to that of the much larger official Pythia-160M model.

3.3 Downstream Tasks Evaluation

We use the PonderingPythia models pretrained in the previous subsection to conduct downstream task evaluations.

3.3.1 General Downstream Tasks

We consider various widely-used benchmarks, including the tasks originally used by Pythia (LAMBADA (Paperno et al., 2016), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2021), ARC-Easy and ARC-Challenge (Clark et al., 2018), SciQ (Welbl et al., 2017). Additionally, we include HellaSwag (Zellers et al., 2019) for commonsense reasoning and RACE (Lai et al., 2017) for reading comprehension.

We evaluate both zero-shot and five-shot learning performance using the LM evaluation harness (Gao et al., 2023). Detailed results are shown in Table 3. Across all evaluated model sizes, PonderingPythia consistently and significantly outperforms the official Pythia models, as well as comparable OPT, GPT-Neo, and Bloom models. Remarkably, with only 1/10 of the training data (300B tokens) and fewer parameters (1B vs. 1.1B), our PonderingPythia-1B achieves results comparable to, or even surpassing, TinyLlama—which uses a more advanced LLaMA architecture and 3T tokens. PonderingPythia-1.4B also achieves performance on par with models twice its size.

[4]To match the training FLOPs of vanilla Pythia-70M with PonderingPythia-70M, we trained the vanilla model for 4 epochs.

Table 3: Five-shot and zero-shot evaluations on downstream NLP tasks. All pretrained model weights used for comparison are obtained from their official repositories. We refer to PonderingPythia as Ponder for simplicity. Δacc is compared to the official Pythia models.

Model (#training tokens)	Lambada OpenAI	ARC -E	Lambada Standard	ARC -C	Wino Grande	PIQA	Hella Swag	SciQ	RACE	Avg acc / Δacc ↑
5-shot										
Pythia-70M (300B)	11.9	36.7	9.2	17.1	50.5	58.7	26.7	57.8	25.1	32.6
Ponder-70M (300B)	**28.4**	**44.6**	**21.1**	**18.6**	**52.2**	**62.4**	**28.4**	**75.7**	**26.9**	**39.8** / +7.2
Pythia-160M (300B)	24.9	44.7	19.0	18.4	50.4	63.5	28.6	76.4	27.8	39.3
OPT-125M (300B)	29.8	42.3	26.4	19.7	51.3	63.4	29.0	78.8	30.1	41.2
GPTneo-125M (300B)	29.3	43.9	23.0	19.1	**51.9**	63.9	28.5	79.9	27.9	40.8
Ponder-160M (300B)	**39.9**	**50.1**	**31.2**	**22.1**	51.2	**65.8**	**31.9**	**85.4**	**30.1**	**45.3** / +6.0
Pythia-410M (300B)	43.9	54.7	32.8	22.3	53.4	68.0	33.8	88.9	30.4	47.6
OPT-350M (300B)	38.3	45.4	32.1	20.5	53.0	65.8	31.9	85.7	29.5	44.7
Bloom-560M (366B)	29.4	50.2	29.7	21.9	52.7	64.2	31.4	88.0	30.0	44.2
Ponder-410M (300B)	**48.9**	**58.7**	**43.7**	**26.1**	**54.0**	**70.5**	**37.3**	**91.0**	**32.4**	**51.4** /+3.8
Pythia-1B (300B)	48.3	58.6	35.8	25.4	52.8	71.3	37.7	91.6	31.7	50.4
OPT-1.3B (300B)	54.0	60.4	49.0	26.9	56.9	72.4	38.5	91.8	35.4	52.7
GPTneo-1.3B (300B)	49.9	59.9	44.5	25.9	56.6	71.6	38.6	86.1	34.5	51.9
Bloom-1.1B (366B)	36.3	54.9	37.4	24.9	53.4	67.6	34.8	88.7	33.0	47.9
Ponder-1B (300B)	**57.7**	**63.2**	**52.5**	**28.6**	**58.6**	**73.3**	**41.9**	**93.4**	**36.3**	**56.2** / +5.8
Pythia-1.4B (300B)	54.5	63.1	44.5	28.8	57.1	71.0	40.5	92.4	34.6	54.1
Bloom-1.7B (366B)	42.5	58.8	41.5	26.2	57.7	68.7	37.6	91.9	33.5	50.9
Ponder-1.4B (300B)	**59.2**	**67.5**	**49.9**	**32.4**	**60.4**	**73.5**	**44.2**	**94.3**	**37.1**	**57.6** / +3.5
Tinyllama-1.1B (3T)	53.8	64.8	45.0	31.1	59.4	73.8	44.9	94.0	36.4	55.9
OPT-2.7B (300B)	60.2	64.7	55.0	29.8	62.2	75.1	46.1	93.0	37.5	58.2
GPTneo-2.7B (300B)	56.0	64.0	51.6	30.1	59.6	73.9	42.4	93.3	35.5	56.3
Bloom-3B (366B)	46.2	63.8	47.1	31.7	57.8	70.8	41.4	93.4	34.6	54.1
Pythia-2.8B (300B)	59.0	67.0	50.7	31.0	61.1	74.4	45.3	93.7	35.9	57.6
0-shot										
Pythia-70M (300B)	18.7	36.7	13.5	18.8	51.1	59.9	26.6	59.9	23.9	34.3
Ponder-70M (300B)	**33.5**	**42.5**	**24.1**	**19.1**	**51.1**	**61.5**	**28.3**	**70.3**	**27.8**	**39.8** / +5.5
Pythia-160M (300B)	33.0	43.8	21.4	18.9	52.0	62.2	28.4	73.7	27.9	40.1
OPT-125M (300B)	37.9	43.5	29.0	19.0	50.3	62.8	29.2	75.2	**30.1**	41.9
GPTneo-125M (300B)	37.5	43.9	26.0	19.1	50.4	63.0	28.7	76.5	27.5	41.4
Ponder-160M (300B)	**44.3**	**46.8**	**33.8**	**20.5**	**52.3**	**63.9**	**31.9**	**76.8**	29.6	**44.4** / +4.3
Pythia-410M (300B)	51.4	**52.2**	36.4	21.4	53.8	66.9	33.7	**81.5**	30.9	47.6
OPT-350M (300B)	45.2	44.0	35.8	20.7	52.3	64.5	32.0	74.9	29.8	44.4
Bloom-560M (366B)	34.3	47.5	33.3	22.4	51.5	63.8	31.5	80.3	30.5	43.9
Ponder-410M (300B)	**56.9**	51.9	**45.3**	**22.6**	**56.0**	**68.7**	**37.0**	81.4	**33.8**	**50.4** / +2.8
Pythia-1B (300B)	55.9	56.8	42.0	24.2	52.5	70.5	37.7	83.3	32.7	50.6
OPT-1.3B (300B)	57.9	57.1	**52.5**	23.4	**59.7**	71.8	41.6	84.3	34.3	53.6
GPTneo-1.3B (300B)	57.1	56.2	45.3	23.2	55.0	71.2	38.6	86.1	34.5	51.9
Bloom-1.1B (366B)	42.6	51.5	42.9	23.6	54.9	67.3	34.5	83.6	32.6	48.2
Ponder-1B (300B)	**62.3**	**60.5**	51.9	**27.0**	56.5	**72.2**	**41.8**	**87.4**	**35.4**	**55.0** / +4.4
Pythia-1.4B (300B)	61.6	60.4	49.7	25.9	57.5	70.8	40.4	86.4	34.1	54.1
Bloom-1.7B (366B)	46.2	56.4	44.5	23.7	56.8	68.5	37.5	85.0	33.2	50.2
Ponder-1.4B (300B)	**65.2**	**62.0**	**53.8**	**27.0**	**60.1**	**72.6**	**44.0**	**89.0**	**35.2**	**56.5** / +2.4
Tinyllama-1.1B (3T)	58.8	60.3	49.3	28.0	59.0	73.3	45.0	88.9	36.4	55.4
OPT-2.7B (300B)	63.5	60.8	56.0	26.8	61.2	73.8	45.9	85.8	36.2	56.7
GPTneo-2.7B (300B)	62.1	61.2	51.6	27.5	57.8	72.3	42.7	89.3	35.1	55.5
Bloom-3B (366B)	51.7	59.4	50.9	28.0	58.7	70.8	41.4	88.8	35.2	53.9
Pythia-2.8B (300B)	64.6	64.4	54.3	29.5	60.2	73.8	45.4	88.5	34.9	57.3

3.3.2 Instruction-following Ability Evaluation

To assess the instruction-following capability of our model, we further fine-tuned PonderingPythia-1B and 1.4B, as well as the corresponding official Pythia models, on the Alpaca dataset using the same settings (Taori et al., 2023). The fine-tuned models were evaluated with MT-Bench (Zheng et al., 2023), a popular multi-turn question benchmark. The experimental results are shown in Figure 4. As illustrated, both PonderingPythia-1B and 1.4B consistently outperform their official Pythia counterparts across all subtasks, achieving average improvements of 0.33 and 0.55, respectively.[5]

3.4 Impact of Pondering Steps

To further investigate the effect of pondering steps on model performance, we trained several 160M-parameter Pythia models from scratch with different numbers of pondering steps: 0 (baseline), 1, 3, 5, and 10. These models were pretrained on a 30B-token subset of the Pile dataset. The results, presented in Figure 5, clearly show that increasing the number of pondering steps consistently reduces the language modeling loss on the Pile validation set. This demonstrates the significant potential and scalability of our method, suggesting that model performance can be further improved by increasing the depth of pondering steps.

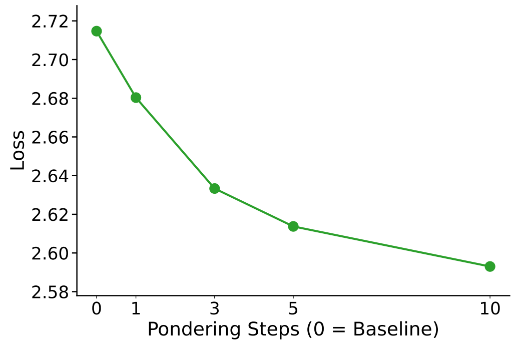


Figure 5: Increasing the number of pondering steps consistently reduces loss on the Pile validation set.

4 Limitations and Future Work

4.1 Limitations

There are two limitations to our work. Firstly, due to computational constraints, we only scaled our method up to a 1.4B-parameter model trained on 300B tokens from the Pile dataset. It would be interesting to extend our approach to larger models and larger-scale datasets in the future. Secondly, although our results demonstrate that the proposed method scales better than vanilla models under the same training FLOPs (Figure 3), it also introduces additional inference overhead (increasing roughly linearly with the number of pondering steps), similar to test-time scaling methods.

4.2 Future Work

There are several promising directions for future work. Firstly, our proposed method is not limited to decoder-only architectures or language modeling; it has the potential to be applied to a wide range of model types and domains. For example, it could be adapted to state-space models such as Mamba (Gu & Dao, 2023), encoder models, or RWKV (Peng et al., 2023), as well as extended to other areas. Another promising direction is the introduction of token-adaptive pondering, which may significantly reduce computation and further enhance our method. It would also be interesting to investigate the interpretability of the pondering process, such as how the model "thinks" during pondering, the semantics of the pondering embedding, and whether the model learns to reflect on its predictions through pondering. Finally, exploring the combination of our method with orthogonal approaches such as CoT and other test-time scaling methods could also be an interesting direction.

5 Related Work

5.1 Test-Time Compute Scaling

Scaling test-time computation has emerged as an influential approach, providing an effective alternative to merely increasing model parameters (Snell et al., 2024). Current methods for enhancing

[5]The marginal gains on Coding and Math tasks may be attributed to the limited coding and mathematical abilities of the Pythia models.

test-time computation primarily fall into two categories: parallel scaling and sequential scaling (Zeng et al., 2024; Muennighoff et al., 2025).

Parallel scaling methods involve generating multiple candidate solutions simultaneously and selecting the best solution based on certain evaluation criteria. Prominent examples of this paradigm include Best-of-N search (Cobbe et al., 2021; Sun et al., 2024; Gui et al., 2024; Amini et al., 2024; Sessa et al., 2024) and Majority Vote (Wang et al., 2022). The primary difference among these parallel approaches lies in their strategy for selecting the optimal outcome. Nevertheless, parallel scaling approaches often encounter difficulty accurately identifying the best candidate solution among multiple generated possibilities (Stroebl et al., 2024; Hassid et al., 2024).

Sequential scaling methods, conversely, focus on progressively refining the model's reasoning through multiple iterative steps. Representative methods include CoT (Wei et al., 2022; Nye et al., 2021), iterative rethinking and revision (Huang et al., 2022; Min et al., 2024; Madaan et al., 2024; Wang et al., 2024b,b; Lee et al., 2025; Hou et al., 2025; Muennighoff et al., 2025; Li et al., 2025). Following the emergence of powerful OpenAI o1 and DeepSeek R1, leveraging extensive chain-of-thought prompting to scale test-time computation has become increasingly popular. Nonetheless, existing methods often come with limitations, including reliance on specially curated datasets (Allen-Zhu & Li, 2023), extended context windows (Zhu et al., 2025), and complex reinforcement learning strategies (Pang et al., 2025). In contrast, our method effectively circumvents these limitations.

5.2 Latent Thinking in Language Models

Latent thinking in LMs typically refers to the hidden computational processes within transformer architectures (Yang et al., 2024; Biran et al., 2024). Prior works exploring latent thinking can generally be divided into two categories based on their implementation methods:

Adding Additional Tokens: Wang et al. (2024a) proposed predicting a planning token as a discrete latent variable prior to generating reasoning steps. Similarly, Pfau et al. (2024) investigated filler tokens (e.g., "...") and found them effective for parallelizable problems, although they also highlighted limitations compared to CoT methods, potentially constraining scalability to more complex reasoning tasks. Zhou et al. (2024) pretrained models by randomly inserting a learnable token into the training corpus, demonstrating improvements across various tasks. Furthermore, Quiet-STaR employs reinforcement learning to train models to generate intermediate rationales at each token, thereby enhancing reasoning capabilities (Zelikman et al., 2024).

Reusing Hidden States: Early Universal Transformers (Csordás et al.) reused hidden states through a transition network, aiming to enhance the encoder-decoder transformer architecture toward Turing completeness. A more recent method(Geiping et al., 2025) iterates over multiple layers of a language model to refine intermediate hidden states, treating this process as a form of latent thinking. Giannou et al. (2023) proposed looped transformer, recycling output hidden states back into input embeddings for algorithmic tasks. Similarly, Hao et al. (2024) fine-tuned models to reason directly within continuous latent spaces, using final hidden states as embeddings to achieve reasoning without explicit CoT. In contrast, our method relies on pondering embeddings derived from the predicted probabilities of LMs.

6 Conclusion

In this paper, we introduce the pondering process into language models through solely self-supervised learning. Pondering LM can be naturally learned through pretraining on large-scale general corpora. Our extensive experiments across three widely adopted architectures—GPT-2, Pythia, and LLaMA—highlight the effectiveness and generality of our proposed method. Notably, our Pondering-Pythia consistently outperforms the official Pythia model on language modeling tasks, scaling curves, downstream tasks, and instruction-following abilities when pretrained on the large-scale Pile dataset. As increasing the number of pondering steps further improves language model performance, we posit that our approach introduces a promising new dimension along which language model capabilities can be scaled.

References

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 3.2, knowledge manipulation. *arXiv preprint arXiv:2309.14402*, 2023.

Allen-Zhu, Z. and Li, Y. Physics of language models: Part 3.3, knowledge capacity scaling laws. *arXiv preprint arXiv:2404.05405*, 2024.

Amini, A., Vieira, T., Ash, E., and Cotterell, R. Variational best-of-n alignment. *arXiv preprint arXiv:2407.06057*, 2024.

Biderman, S., Schoelkopf, H., Anthony, Q. G., Bradley, H., O'Brien, K., Hallahan, E., Khan, M. A., Purohit, S., Prashanth, U. S., Raff, E., et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Biran, E., Gottesman, D., Yang, S., Geva, M., and Globerson, A. Hopping too late: Exploring the limitations of large language models on multi-hop queries. *arXiv preprint arXiv:2406.12775*, 2024.

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

Black, S., Gao, L., Wang, P., Leahy, C., and Biderman, S. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL https://doi.org/10.5281/zenodo.5297715. If you use this software, please cite it using these metadata.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

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

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*, 9, 2021.

Csordás, R., Irie, K., Schmidhuber, J., Potts, C., and Manning, C. D. Moeut: Mixture-of-experts universal transformers. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

DeepSeek-AI, Guo, D., Yang, D., and et al., H. Z. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. 2025. URL https://arxiv.org/abs/2501.12948.

Fedorenko, E., Piantadosi, S. T., and Gibson, E. A. Language is primarily a tool for communication rather than thought. *Nature*, 630(8017):575–586, 2024.

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

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

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Giannou, A., Rajput, S., Sohn, J.-y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Gui, L., Gârbacea, C., and Veitch, V. Bonbon alignment for large language models and the sweetness of best-of-n sampling. *arXiv preprint arXiv:2406.00832*, 2024.

Hackenburg, K., Tappin, B. M., Röttger, P., Hale, S. A., Bright, J., and Margetts, H. Scaling language model size yields diminishing returns for single-message political persuasion. *Proceedings of the National Academy of Sciences*, 122(10):e2413443122, 2025.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Hassid, M., Remez, T., Gehring, J., Schwartz, R., and Adi, Y. The larger the better? improved llm code-generation via budget reallocation. *arXiv preprint arXiv:2404.00725*, 2024.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022a.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., et al. An empirical analysis of compute-optimal large language model training. *Advances in Neural Information Processing Systems*, 35:30016–30030, 2022b.

Hou, Z., Lv, X., Lu, R., Zhang, J., Li, Y., Yao, Z., Li, J., Tang, J., and Dong, Y. Advancing language model reasoning through reinforcement learning and inference scaling. *arXiv preprint arXiv:2501.11651*, 2025.

Huang, J., Gu, S. S., Hou, L., Wu, Y., Wang, X., Yu, H., and Han, J. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

Le Scao, T., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. Bloom: A 176b-parameter open-access multilingual language model. 2023.

Lee, K.-H., Fischer, I., Wu, Y.-H., Marwood, D., Baluja, S., Schuurmans, D., and Chen, X. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*, 2025.

Li, D., Cao, S., Griggs, T., Liu, S., Mo, X., Patil, S. G., Zaharia, M., Gonzalez, J. E., and Stoica, I. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*, 2025.

Li, L. H., Hessel, J., Yu, Y., Ren, X., Chang, K.-W., and Choi, Y. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2665–2679, Toronto, Canada, jul 2023. Association for Computational Linguistics.

Li, W., Liu, X., Li, Y., Jin, Y., Tian, H., Zhong, Z., Liu, G., Zhang, Y., and Chen, K. Understanding communication characteristics of distributed training. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, pp. 1–8, 2024.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Min, Y., Chen, Z., Jiang, J., Chen, J., Deng, J., Hu, Y., Tang, Y., Wang, J., Cheng, X., Song, H., et al. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems. *arXiv preprint arXiv:2412.09413*, 2024.

Muennighoff, N., Rush, A., Barak, B., Le Scao, T., Tazi, N., Piktus, A., Pyysalo, S., Wolf, T., and Raffel, C. A. Scaling data-constrained language models. *Advances in Neural Information Processing Systems*, 36:50358–50376, 2023.

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candès, E., and Hashimoto, T. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Pang, B., Dong, H., Xu, J., Savarese, S., Zhou, Y., and Xiong, C. Bolt: Bootstrap long chain-of-thought in language models without distillation. *arXiv preprint arXiv:2502.03860*, 2025.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Pati, S., Aga, S., Islam, M., Jayasena, N., and Sinclair, M. D. Computation vs. communication scaling for future transformers on future hardware. *arXiv preprint arXiv:2302.02825*, 2023.

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., Cao, H., Cheng, X., Chung, M., Grella, M., et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Pfau, J., Merrill, W., and Bowman, S. R. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 2024.

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

Sessa, P. G., Dadashi, R., Hussenot, L., Ferret, J., Vieillard, N., Ramé, A., Shariari, B., Perrin, S., Friesen, A., Cideron, G., et al. Bond: Aligning llms with best-of-n distillation. *arXiv preprint arXiv:2407.14622*, 2024.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Stroebl, B., Kapoor, S., and Narayanan, A. Inference scaling flaws: The limits of llm resampling with imperfect verifiers. *arXiv preprint arXiv:2411.17501*, 2024.

Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Sun, H., Haider, M., Zhang, R., Yang, H., Qiu, J., Yin, M., Wang, M., Bartlett, P., and Zanette, A. Fast best-of-n decoding via speculative rejection. *arXiv preprint arXiv:2410.20290*, 2024.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Villalobos, P., Sevilla, J., Heim, L., Besiroglu, T., Hobbhahn, M., and Ho, A. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *arXiv preprint arXiv:2211.04325*, 1, 2022.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wang, X., Caccia, L., Ostapenko, O., Yuan, X., Wang, W. Y., and Sordoni, A. Guiding language model reasoning with planning tokens, 2024a. URL https://arxiv.org/abs/2310.05707.

Wang, Y., Wu, Y., Wei, Z., Jegelka, S., and Wang, Y. A theoretical understanding of self-correction through in-context alignment. *arXiv preprint arXiv:2405.18634*, 2024b.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Welbl, J., Liu, N. F., and Gardner, M. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

Yang, S., Gribovskaya, E., Kassner, N., Geva, M., and Riedel, S. Do large language models latently perform multi-hop reasoning? *arXiv preprint arXiv:2402.16837*, 2024.

Yue, Y., Chen, Z., Lu, R., Zhao, A., Wang, Z., Song, S., and Huang, G. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Zelikman, E., Harik, G., Shao, Y., Jayasiri, V., Haber, N., and Goodman, N. D. Quiet-star: Language models can teach themselves to think before speaking, 2024. URL https://arxiv.org/abs/2403.09629.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zeng, Z., Cheng, Q., Yin, Z., Wang, B., Li, S., Zhou, Y., Guo, Q., Huang, X., and Qiu, X. Scaling of search and learning: A roadmap to reproduce o1 from reinforcement learning perspective. *arXiv preprint arXiv:2412.14135*, 2024.

Zhang, B. and Sennrich, R. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Zhou, J., Pang, L., Shen, H., and Cheng, X. Think before you speak: Cultivating communication skills of large language models via inner monologue, 2024. URL https://arxiv.org/abs/2311.07445.

Zhu, D., Wei, X., Zhao, G., Wu, W., Zou, H., Ran, J., Wang, X., Sun, L., Zhang, X., and Li, S. Chain-of-thought matters: Improving long-context language models with reasoning path supervision. *arXiv preprint arXiv:2502.20790*, 2025.