Inner Thinking Transformer: Leveraging Dynamic Depth Scaling to Foster Adaptive Internal Thinking

Yilong Chen[1,2], Junyuan Shang[3‡], Zhenyu Zhang[3], Yanxi Xie[4], Jiawei Sheng[1], Tingwen Liu[1,2†], Shuohuan Wang[3], Yu Sun[3], Hua Wu[3], Haifeng Wang[3]

[1] Institute of Information Engineering, Chinese Academy of Sciences

[2] School of Cyber Security, University of Chinese Academy of Sciences

[3] Baidu Inc.

[4] School of Artificial Intelligence, Beijing Normal University

{chenyilong, shengjiawei, liutingwen}@iie.ac.cn

{shangjunyuan, zhangzhenyu07, wangshuohuan, sunyu02}@baidu.com

Abstract

Large language models (LLMs) face a performance ceiling as scaling parameters becomes impractical. Our observations indicate that while simple tokens are efficiently resolved in early layers with stable gradients, complex tokens trigger abrupt gradient spikes across layers, underscoring architectural limitations. Existing step-by-step reasoning methods, such as Chain-of-Thought, are hindered by their dependence on accurately generating critical tokens. We introduce Inner Thinking Transformer (ITT)—a straightforward approach that enables models to "think" more deeply about important tokens by dynamically assigning extra inference steps through a token-wise dynamic depth architecture with residual iterative reasoning and step encoding. Experiments on LLaMA2-7B models at 355M, 1B, and 3B scales show that ITT consistently outperforms vanilla Transformers, with a 355M ITT model matching the performance of a 1B Transformer, offering a scalable, architecture-aware strategy to enhance LLM reasoning capabilities.

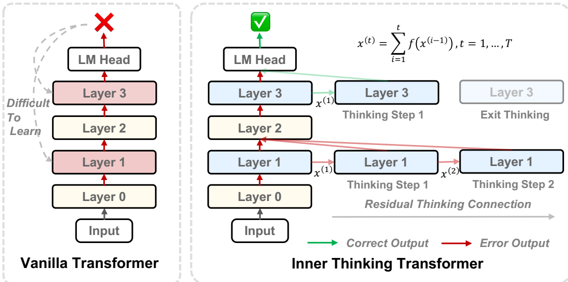

Figure 1: The Transformer, constrained by a limited number of parameters, tends to make errors on difficult samples. We treat each single computation in the model's layers as one step of inner thinking. By training the model to allocate more inner thinking steps at specific layers and organize thinking results, the model can achieve better results without scaling parameters.

1 Introduction

Large language models (LLMs) (Anthropic, 2023; OpenAI, 2023; Touvron et al., 2023) have demonstrated remarkable performance across numerous natural language tasks. Recent studies (Fernandez et al., 2024; Hoffmann et al., 2022; Chen et al., 2024a) indicate that scaling laws for LLM parameters exhibit diminishing returns under constrained data availability and computational resource budgets. Scaling model parameters increases computational and deployment costs, making high-performance models impractical for resource-constrained environments. Meanwhile, smaller models encounter performance bottlenecks primarily attributable to limited parameter space.

Recent approaches, such as Test-Time Scaling ("Slow-Thinking") (Muennighoff et al., 2025; Snell et al., 2024; Ma et al., 2024; Zhang et al., 2025b,a), aim to enhance performance by allocating more computation during the inference search process. While effective, these methods are limited by the reliance on accurately generating key tokens, which can lead to catastrophic reasoning failures (Chen et al., 2023a; Singh et al., 2024; Jiang et al., 2024b), especially in smaller models. Some works enhance model performance through layer sharing (Li et al., 2024b,c), recursion (Ng and Wang, 2024; Dehghani et al., 2019b; Geiping et al., 2025), or implicit reasoning (Deng et al., 2023; Shalev et al., 2024), but they fail to flexibly improve the model's reasoning ability on key tokens, which either suffer from insufficient performance or redundant overhead.

In this work, we aim to explore *how the model can allocate more computation to individual tokens, enhancing testing performance without increasing parameters.* Through analysis in Section 2, we explore how models learn and reason about critical tokens. Our findings reveal that *simple tokens are resolved efficiently in early layers with stable*

†Corresponding author. ‡ Project lead.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28241–28259
July 27 - August 1, 2025

low-gradient flows, while complex tokens cause difficulties across layers, with sudden gradient spikes indicating architectural or parametric issues. The differentiated properties of layers inspire us to propose a novel perspective on the model's internal reasoning process: *Inner Thinking*. Inner thinking conceptualizes the evolution of hidden states layer by layer, with each layer representing a distinct implicit reasoning step for deriving a single token.

Intuitively, we can extend and combine multiple inner thinking steps to break the model's performance bottleneck. Therefore, we propose a novel approach called Inner Thinking Transformer (ITT). ITT enhances token-level reasoning by dynamically allocating additional thinking steps to key tokens and iteratively accumulating residual thinking results to refine tokens' representations. As shown in Figure 1, the model learns to "think" more deeply on important information during training. Specifically, we design a dynamic token-wise depth architecture based on *Adaptive Token Routing* networks and adopt a *Residual Thinking Connection* mechanism (RTC) that gradually converges toward better outcomes at each step. In addition, we introduce a *Thinking Step Encoding* scheme to better differentiate between successive thinking steps.

Notably, while trained under specific thinking settings, our architecture can *flexibly allocate more computational resources during testing time* to improve performance or achieve a balanced tradeoff between resources and performance (see Figure 6). *The routing network autonomously develops a thinking pattern that strategically balances depth and breadth*: specific thinking steps are allocated for intensive processing of complex tokens, while more efficient pathways handle simpler tokens (see Figure 7). In general, ITT mitigates the performance bottleneck in reasoning for individual tokens and can be combined with COT methods to resolving reasoning challenges for critical tokens.

Experimentally, we construct both vanilla Transformer, Loop variants and ITT variants across three scales (162M, 230M, and 466M parameters) following the LLaMA architecture. Evaluated on an 11-task benchmark, ITT consistently outperforms Transformer and Loop variants with an equivalent parameters. ITT achieves higher performance with the same FLOPs and saves 43.2% of the training data budget compared to Transformer. Notably, the ITT ×4 -162M model significantly surpasses the 230M Transformer and even achieves 96.5% performance of 466M Transformer. Overall, ITT introduces an inherent test-time scaling in the model, achieving both performance and efficiency balance through its elastic deep computation paradigm.

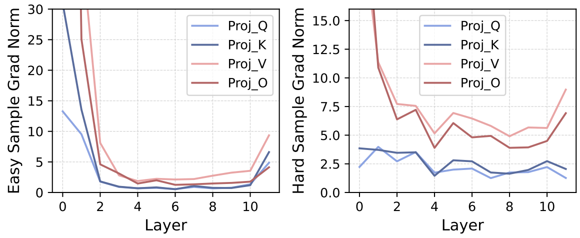

Figure 2: Layer's Gradient Nuclear Norm of the Attention matrices of GPT-2 on hard or simple samples.

2 Observation

To investigate how models learn about critical tokens, our empirical analysis of GPT-2's attention matrices through gradient nuclear norm (GNN) measurements (Li et al., 2024a) reveals systematic patterns in layer-wise dynamics. Using the AQuA corpus (Ling et al., 2017), we firstly train GPT-2 in 100 samples then categorize samples in evaluation into easy (model answers correctly) and hard (model answers incorrectly). In Figure 2, for easy samples, GNN values decay exponentially across early layers (L0-L2) and final layers (L11), stabilizing below 3 in layers (L3-L10). In contrast, hard samples exhibit persistent GNN oscillations throughout all 12 layers, punctuated by abrupt spikes at strategic layer positions (L3, L5, L7, L9).

These observations reveal one of the underlying reasons for the presence of hard-to-learn samples in models: as shown in Figure 1, certain parameters face significant optimization difficulties due to architectural limitations (e.g., insufficient depth) or parameter constraints. Many studies suggest that Transformer layers exhibit unique functional characteristics and training variances (Alizadeh et al., 2024; Sun et al., 2025; Takase and Kiyono, 2023).. This inspires us to propose a framework where *each layer transformation in the model is viewed as a single thinking step on latent information.* By studying the inner thinking process, we aim to design corresponding architectures to optimize model's learning difficulty and inference performance.

3 Method

In this section, we introduce our Inner Thinking (ITT) framework (Figure 3) to enhance transformer models by dynamically deepening token-level reasoning. We begin in Section 3.1 by formalizing inner thinking steps within the transformer. Section 3.2 then details the Residual Thinking Connection, where inner steps are extended via residual accumulation. In Section 3.3, we present the Adaptive Token Routing, which employs a weight predictor to select the most critical tokens for further thinking. Finally, Section 3.4 demonstrate how ITT enhances learning efficiency in backporpogation.

3.1 Inner Thinking Step in Transformer

Traditional reasoning in Transformer models typically relies on token-by-token generation. Given an input x, the output sequence $y = (y_1, y_2, \ldots, y_N)$ is generated as

$$P(y \mid x) = \prod_{n=1}^{N} P(y_n \mid y_{<n}, x), \tag{1}$$

However, errors in key tokens can propagate, potentially leading to an incorrect result. To investigate the intrinsic mechanisms in single-token generating, we propose a novel concept of *Inner Thinking* in model's depth that decomposes the generation of each token into a series of internal thinking steps. Specifically, given an initial state $x^{(0)}$, we define Inner Thinking as

$$X^{(t)} = f^{(t)}\big(x^{(t-1)}\big), \quad t = 1, 2, \ldots, T, \tag{2}$$

where $f^{(t)}(\cdot)$ represents the transformation corresponding to the t-th thinking step (consist of one or more Transformer layers) and T is the maximum number of steps. The final token is then generated based on the output of the last thinking step:

$$P(y \mid x) = \mathrm{softmax}\big(W\, x^{(T)} + b\big), \tag{3}$$

with W and b denoting the weights and bias for the output projection. Define $\mathcal{L}(\cdot, y)$ measures the discrepancy between final state $X^{(T)}$ and the target token y, we have two scenarios:

Early Exit: If at an intermediate step $t_0 < T$, the state $x^{(t_0)}$ is close enough to the target (i.e., $\mathcal{L}\big(x^{(t_0)}, y\big) < \epsilon$, where ϵ is a threshold), the model can stop and output the token as $y = \psi\big(x^{(t_0)}\big)$, where $\psi(\cdot)$ is the decoding function. This allows the model to achieve correct results with fewer Inner Thinking Steps, improving efficiency.

Performance Deficiency: Conversely, if even after all T internal steps the discrepancy remains high (i.e., $\mathcal{L}\big(x^{(T)}, y\big) > \epsilon$), it indicates that the Inner Thinking was insufficient to correctly approximate the target. This scenario highlights potential areas for improvement in the model's reasoning capacity or its internal step design.

3.2 Residual Thinking Connection

Under the framework defined in Section 3.1, we aim to enhance the model's performance to reduce **Performance Deficiencies**. For challenging examples, high gradient values are observed in Section 2, indicating that the model faces optimization difficulties. To address these issues, a natural approach is to increase the number of inner thinking steps in one layer's computation. Therefore, we propose a *Residual Thinking Connection* (RTC) mechanism that train model's layer parameters to **learn iterative thinking capabilities**, reducing the difficulty of single-step thinking and enabling multiple uses of parameters to break performance bottlenecks.

Let $x^{(0)} \in \mathbb{R}^d$ denote RTC Layer input of a token representation, where d is the hidden dimension. We denote $f : \mathbb{R}^d \to \mathbb{R}^d$ as the layer transformation, T is the maximum number of thinking steps. In RTC, the final output after t iterative steps is computed by cumulatively accumulating each step's outputs:

$$x^{(t)} = \sum_{i=1}^{t} \Big(f\big(x^{(i-1)}\big) \odot \phi^{(i)} \Big), t = 1, \ldots, T, \tag{4}$$

where $\phi^{(t)} \in \mathbb{R}^d$ the learnable thinking position encoding associated with the t-th inner thinking step, which measuring the differences and importance of each step. Rather than processing the input representation only once, RTC Layer iteratively refine it by adding the residual contributions of each step's layer-output together with a learnable encoding. Compared to direct looping (Ng and Wang, 2024; Dehghani et al., 2019b), *RTC not only enables deeper thinking but also effectively measures and combines each thinking step, allowing them to complement each other.* RTC provides the foundation for scaling Inner Thinking during testing.

3.3 Adaptive Token Routing

RTC in Section 3.2 provides a method to enhance inner thinking. However, *different tokens require a varying number of thinking steps in the model*, as show in Section 2. Moreover, we aim for the model to learn detailed, task-specific information at each step. To avoid unnecessary computation and information interference from processing all tokens at once, we introduce Adaptive Token Routing (ATR). Inspired by deep conditional computation (Raposo et al., 2024; Zhang et al., 2024), ATR, based on a routing network, selects the most important tokens for thinking at each step.

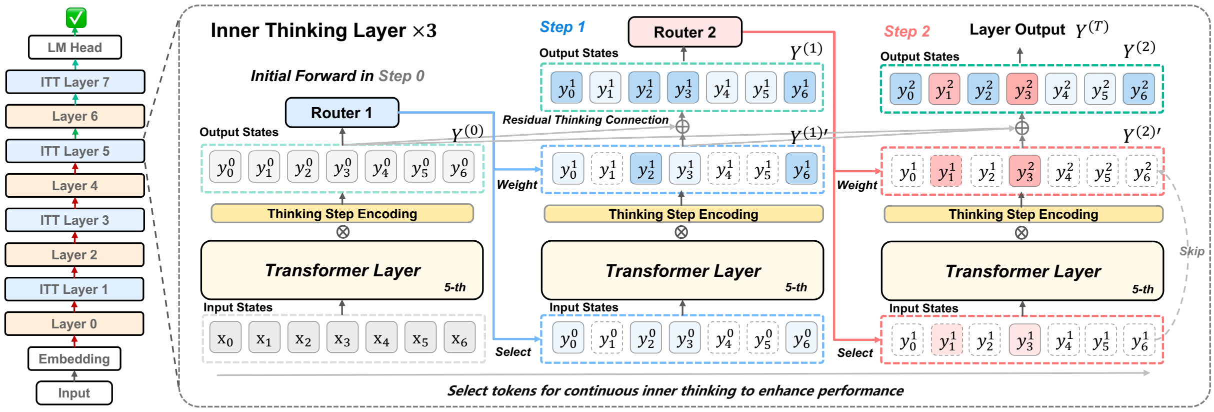

Figure 3: An illustration of ITT: ITT uses Adaptive Token Routing to select and weight important tokens for each inner thinking step. Based on Thinking Step Encoding and Residual Thinking Connection, ITT layer iterates thinking multiple times, accumulating each step's results for improved layer output.

Let the input sequence be denoted by $X \in \mathbb{R}^{n\times d}$, where n is the sequence length. We perform a forward pass to obtain the output $Y^{(0)} \in \mathbb{R}^{n\times d}$, and then linear weight predictor $\mathcal{R}^{(0)} \in \mathbb{R}^{d\times 1}$ is applied to $Y^{(0)}$ to generate an importance score:

$$Y^{(0)} = f(X), \quad w^{(1)} = \mathcal{R}^{(1)}(Y^{(0)}) \in \mathbb{R}^{n}, \tag{5}$$

and we denote by $P_\rho(w^{(1)})$ the ρ-th percentile of these scores, with ρ being a predefined selection ratio. For a given thinking step t, the calculation process in ITT layer can be formulated as:

$$Y_i^{(t)\prime} = \begin{cases} \alpha^{(t)} w_i^{(t)} f\left(Y_i^{(t-1)}\right), & \text{if} \quad w_i^{(t)} > P_\rho(w^{(t)}), \\ Y_i^{(t-1)}, & \text{if} \quad w_i^{(t)} \le P_\rho(w^{(t)}), \end{cases} \tag{6}$$

where $\alpha^{(t)}$ is a hyperparam in t step, $w_i^{(t)} > P_\rho(w^{(t)})$ is the indicator function selecting only the tokens with predicted weights exceeding the threshold. The router $\mathcal{R}^{(t)}$*modulates the decision to execute an additional thinking iteration based on the current token representation and the step-specific encoding.* For tokens deemed important, the model applies an extra weighted transformation. Conversely, tokens that do not meet the selection criteria bypass the extra processing, preserving their previous representation. The router's weights are part of the gradient path, allowing the routing parameters to be updated through backpropagation.

Finally, ITT (in Figure 3) combine the results of each step using RTC, following Equation 4:

$$Y^{(t)} = Y^{(0)} \odot \phi^{(0)} + \sum_{i=1}^{t} \left(Y_i^{(i)\prime} \odot \phi^{(i)} \right), \tag{7}$$
$$t = 1, \ldots, T.$$

This unified update thus integrates RTC with dynamic, token-level routing, enabling the model to *adaptively allocate computational resources only where deeper thinking is required.* By iteratively selecting a subset of tokens for deeper processing, the model can efficiently reinforce key tokens without increasing the model parameter. In practice, the ITT layer can be flexibly improved based on the model layers. We insert the ITT layer at regular intervals alongside the model's original layers to construct a flexible inner thinking model, and optimize all model parameters using the language modeling cross-entropy loss: $\mathbb{L} = \mathbb{L}_{\text{CE}}$.

3.4 Optimization

In this section, we prove Residual Thinking Learning *extends single-step optimization into multi-step optimization, making it easier to converge* during backpropagation compared to a direct one-step mapping. Let $y^* \in \mathbb{R}^d$ the corresponding ground-truth, Θ' represents the origin Layer parameters, and θ represents th ITT layer parameters. The optimization objective is to minimize the loss:

$$\mathcal{L}\big(F(x;\Theta',\theta), y^*\big) = \mathcal{L}\big(G(f_T(x;\theta);\Theta'), y^*\big). \tag{8}$$

For each step's parameter θ, the gradient is computed using the chain rule:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial Y^{(t)}} \cdot \prod_{j=t+1}^{T} \left[I + \frac{\partial \Delta_j(Y^{(j)};\theta)}{\partial Y^{(j)}} \right] \cdot \frac{\partial \Delta_k(Y^{(0)};\theta)}{\partial \theta}. \tag{9}$$

Assuming that the corrections Δ_j are small, we can approximate the product term by the identity matrix I, yielding:

$$\frac{\partial \mathcal{L}}{\partial \theta} \approx \frac{\partial \mathcal{L}}{\partial Y^{(t)}} \cdot \frac{\partial \Delta_k(Y^{(0)};\theta)}{\partial \theta}. \tag{10}$$

This shows that the gradient update at each small step is nearly equal to the global gradient multiplied by the derivative of the local mapping, aligning with global loss reduction. Assuming each iteration reduces the error by a factor of c, this leads to exponential decay c^t, proving that iterative corrections ensure stable, efficient convergence. In summary, our method avoids excessive scaling or distortion from deep chain propagation. It extends single-step optimization to multi-step, easing convergence and preventing gradient vanishing or explosion.

4 Experiments

4.1 Setup

Data. To pretrain ITT models and baseline models, we employ the RedPajama (TogetherAI, 2023), which parallels the LLaMA training data across seven domains: CommonCrawl, C4, GitHub, Wikipedia, Books, ArXiv, and Stack-Exchange. This dataset comprises a 2 million tokens validation set and a 50 billion tokens training set.

Training. Our experimental framework utilizes the Sheared-LLaMA codebase (Xia et al., 2023) implemented on the Composer package (Team, 2021), and is executed on 8 NVIDIA A100 GPUs (80GB). The models are trained with a sequence length of 4096, employing a global batch size of 256. ITT models are trained for 50000 steps (50B token budget). The learning rates were set at 3e-4 for all parameters. The baselines and all ITT models follow the same training setup, starting from random initialization and training on the same dataset.

Evaluation. We employed the lm-evaluation-harness (Gao et al., 2023) to evaluate our models. For common sense and reading comprehension tasks, we report 0-shot accuracy for SciQ (Welbl et al., 2017), PIQA (Bisk et al., 2020), WinoGrande (WG) (Sakaguchi et al., 2020), ARC Easy(ARC-E) (Clark et al., 2018a), and 10-shot HellaSwag (Hella.) (Zellers et al., 2019), alongside 25-shot accuracy for ARC Challenge (ARC-C) (Clark et al., 2018b). For continued QA and text understanding, we report 0-shot accuracy for LogiQA (Liu et al., 2020), 32-shot BoolQ (Clark et al., 2019), and 0-shot LAMBADA (Lam.) (Paperno et al., 2016). All reported results are calculated with the mean and stderr of multiple experiments.

Baseline. Following the architecture of LLaMA2, we constructed models at three parameter scales: 162M, 230M, and 466M, with hidden dimensions of 1024, 1536, and 2048, as shown in Table 5. For each parameter scale, we develop three variants:

- The Loop Neural Network design (Ng and Wang, 2024; Dehghani et al., 2019b; Geiping et al., 2025) implements model-level recurrence for iterative refinement.
- Our ITT architecture, adaptively selecting a subset of tokens for deeper thinking.

We experiment with three thinking step scaling factors—$2\times$, $3\times$ and $4\times$. We replace every other layer of original model with a Loop or ITT layer.

4.2 Result

Foundational Capabilities. Table 1 shows the performance improvements of ITT (pink) and Loop (blue) on LLaMA 2's 162M, 230M, and 466M versions. Both methods enhance model performance by increasing computational allocation during training and inference without expanding parameters. **Thanks to its unique RTC design, ITT achieves better test-time scaling performance** than Loop, as shown in Figure 5. For example, the 162M ITT $\times 4$ configuration improves the baseline by 1.7% with 4-step deep thinking in 50% of layers, while Loop improves only by 0.3% after 4 iterations. **The advantages of ITT become clearer as model scale increases**, with improvements of 1.7%, 2.1%, and 1.7% for the 162M, 230M, and 466M models. ITT shows overall enhancement across nearly all metrics, with notable improvements in ARC-E, BoolQ, and LAMBADA, reflecting gains in generative and reasoning abilities.

Convergence. Figure 4 Left and Middle visualize the training loss and eval perplexity during 50B-token pre-training for LLaMA 2-2162M, Loop$\times$4, and ITT $\times$4. **ITT demonstrates superior training stability and efficiency**, with smoother, lower perplexity trajectories compared to LLaMA 2-230M and Loop. Notably, ITT $\times$4 shows a 0.09 loss reduction compared to baseline and 0.4 to Loop at 50B tokens. **ITT also reveals remarkable data efficiency**: it matches LLaMA 2-162M's performance using only 56.8% of the training data, showcasing its capability in parameter-efficient scaling and data-efficient learning.

Computational Efficiency. As shown in Figure 4 (Right), Figure 6 (Left), and Table 1, **ITT maintains high computational efficiency during test-time scaling**. With 3-step deep thinking, ITT incurs only 84% of Loop's computational cost, dropping to 70% at 4 steps. Remarkably, **ITT**

Model-Params	FLOPs	Commonsense & Reading Comprehension						Continued		LM	Knowledge	Avg.
		SciQ	PIQA	WG	ARC-E	ARC-C	Hella.	LogiQA	BoolQ	Lam.	MMLU	
LLaMA2-162M	1.88	72.0	62.7	51.9	41.7	19.2	28.8	24.0	50.3	28.6	25.2	40.4
Loop ×3-162M	3.76	71.8	63.1	53.0	40.4	19.1	29.1	20.9	51.9	28.8	25.7	40.4
Loop ×4-162M	4.70	72.8	62.4	52.6	41.8	19.8	29.4	22.0	49.9	30.1	26.3	40.7
ITT ×2 -162M	2.72	72.1	63.5	52.1	41.1	19.2	29.1	21.4	51.4	29.2	25.5	40.6
ITT ×3 -162M	3.19	73.9	62.5	50.6	43.6	19.3	29.2	20.6	52.1	37.1	25.8	41.5
ITT ×4 -162M	3.29	72.4	63.9	52.3	43.4	20.5	29.3	22.8	56.8	33.9	26.0	42.1
LLaMA2-230M	2.87	72.8	65.0	49.3	44.0	19.9	29.1	20.6	60.2	31.7	25.5	41.8
Loop ×3-230M	3.59	71.1	64.3	51.5	41.7	20.3	30.2	22.6	61.2	33.5	26.4	42.3
Loop ×4-230M	3.95	74.1	65.1	52.0	41.7	20.1	30.2	18.6	61.0	32.5	26.7	42.2
ITT ×2 -230M	3.19	72.7	64.6	52.2	43.3	20.5	29.7	22.0	59.7	32.6	25.9	42.3
ITT ×3 -230M	3.37	74.3	65.7	52.8	44.9	20.8	30.8	23.1	62.5	34.2	26.3	43.5
ITT ×4 -230M	3.41	75.1	66.2	53.5	45.0	21.1	31.2	22.4	62.7	34.8	26.6	43.9
LLaMA2-466M	4.92	75.5	66.5	51.5	45.2	20.4	31.3	21.2	62.6	36.6	25.4	43.6
Loop ×3-466M	6.15	74.3	65.8	52.9	44.0	21.0	32.0	22.5	59.2	37.2	26.1	43.5
Loop ×4-466M	6.77	76.8	67.0	50.7	46.5	20.9	32.2	20.1	59.0	40.1	24.8	43.8
ITT ×2 -466M	5.47	75.9	66.2	52.7	45.4	21.2	32.1	21.8	60.7	38.4	25.7	43.9
ITT ×3 -466M	5.78	77.9	66.4	53.7	46.7	22.0	32.8	22.6	59.1	39.3	26.7	44.7
ITT ×4 -466M	5.84	77.2	67.1	54.3	47.3	22.4	32.3	22.7	61.9	40.8	27.0	45.3

Table 1: Comprehensively evaluate the basic capabilities of models with different activated parameters. In particular, ITT ×4-162M represents a model with 162M total parameters using ITT to think total 4 steps.

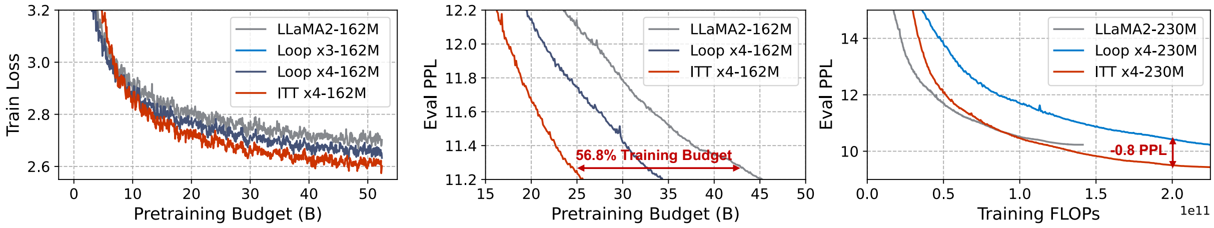

Figure 4: **Left:** Loss curves for 162M-models pre-trained on 50B tokens. **Middle:** Eval Perplexity curves for 162M-models pre-trained on 50B tokens. **Right:** Eval Perplexity for 230M-models with Training FLOPs.

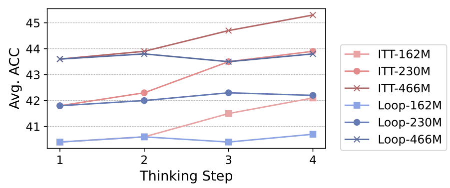

Figure 5: Average accuracy after training 50B tokens for the ITT and Loop models (162M, 230M, 460M) under different thinking step configurations.

outperforms Loop with fewer computational FLOPs, achieving performance similar to models with more parameters. Our experiments show that ITT ×2 outperforms Loop×3 while using only 72% of the computation and exceeds the 230M Dense model with just 70.4% of the parameters. These results highlight the substantial computational efficiency gains from token-wise selective inner thinking in the ITT framework.

Elastic Thinking. Our experiments show that **ITT models can elastically allocate computations for inner thinking.** As seen in Table 2, with 4-step thinking and 70% token participation during training, we can flexibly adjust token selections to enhance performance (e.g., 10.21 PPL in the 70%, 70%, 90% setting, 0.31 PPL lower than the training config), or reduce token selections to lower costs with no performance loss (e.g., 10.47 PPL in the 50%, 50%, 50% setting). We can even *remove a thinking step while maintaining near-identical results* to the training configuration. Figure 6 Left shows the FLOPs and Eval PPL of ITT's elastic inference. Compared to the baselines, ITT achieves a performance-efficiency balance, with the dashed line illustrating **the near-linear tradeoff trend of ITT during testing**. ITT's elastic thinking enables flexible deployment in diverse scenarios.

4.3 Ablation Studies

In Table 3, we compare the ablation results of ITT ×4 with 162M parameters to the baseline under

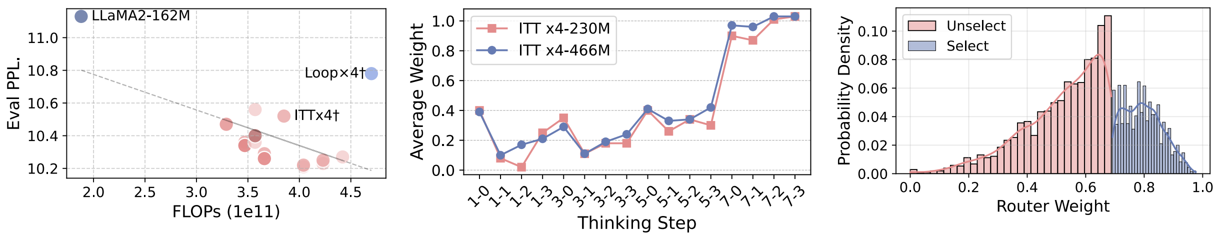

Figure 6: **Left:** Perplexity vs. FLOPs for different selection strategies. Lower left region indicates better performance-efficiency balance. **Middle:** The average weights by the learned Thinking Step Encoding in the ITT x4 model (230M, 460M) across different thinking steps. **Right:** 3-2 step's Router Weight Distribution in ITT ×4.

Method - Select Ratio in Steps	**FLOPs**	**Perplexity ↓**
LLaMA2-162M	1.88	11.13
ITT ×4 - 90%, 90%, 90%	4.42	10.27 (-0.86)
ITT ×4 - 90%, 90%, 0%	3.57	10.40 (-0.73)
ITT ×4 - 90%, 0%, 90%	3.57	10.36 (-0.77)
ITT ×4 - 0%, 90%, 90%	3.57	10.56 (-0.57)
ITT ×4 - 90%, 70%, 90%	4.23	10.23 (-0.90)
ITT ×4 - 70%, 70%, 90%	4.04	10.21 (-0.92)
ITT ×4 - 70%, 70%, 70%†	3.85	10.52 (-0.61)
ITT ×4 - 70%, 70%, 50%	3.66	10.26 (-0.87)
ITT ×4 - 70%, 50%, 50%	3.47	10.34 (-0.79)
ITT ×4 - 50%, 50%, 50%	3.29	10.47 (-0.66)
Loop×4 - 100%, 100%, 100%†	4.70	10.78 (-0.35)

Table 2: Eval Perplexity with different token selection ratios for extended 3-steps thinking. † refers to the model's training configuration.

Method	**FLOPs**	**Perplexity ↓**
ITT ×4 -162M	3.29	10.25
w/o Residual Thinking Connection	3.29	11.02 (+0.77)
w/o Adaptive Token Routing	4.70	10.44 (+0.19)
w/o Thinking Position Encoding	3.29	10.56 (+0.22)
Router Sampling (Top-K)	3.29	10.25 (-)
Router Sampling (Top-P)	3.29	10.34 (+0.09)
Router Weight Norm (Sigmoid)	3.29	10.25 (-)
Router Weight Norm (Tanh)	3.29	10.38 (+0.13)
Token Reweighting (Only Select)	3.29	10.25 (-)
Token Reweighting (Symmetric)	3.29	10.41 (+0.16)
LLaMA2-162M	1.88	11.13 (+1.36)

Table 3: Eval Perplexity with ablation on ITT ×4 -162M. "w.o." indicates the method was ablated.

zero-shot pretraining on 50B tokens, based on Eval PPL. The specific analysis is as follows:

Residual Thinking Connection. Removing this core mechanism causes the largest performance drop (+0.77 PPL), validating our hypothesis about multi-step reasoning. The residual accumulation enables iterative refinement of token representations, particularly crucial for processing linguistically complex patterns. Without RTC, the model may also lose the ability for elastic computation.

Thinking Position Encoding. Thinking Position Encoding provides the model with key information for each thinking step. As shown in Table 3, removing it results in +0.31 PPL., as model loses information about importance of each thinking step.

Adaptive Token Routing. Disabling the dynamic routing mechanism results in a moderate PPL. increase (+0.19), but significantly impacts computational efficiency. This demonstrates the router's dual role: while maintaining prediction quality through selective processing, it achieves more than 50% FLOPs reduction by focusing computation on 50% most critical tokens in each step.

Router Setting. Our experiments validate three critical design choices: The RTC design of ITT relies on explicit token selection signals (e.g., a 0.5 threshold in Sigmoid) for error correction and progressive disambiguation. The cumulative probability characteristic of Top-P conflicts with this deterministic routing mechanism, leading to a disruption in the iterative processing chain of key tokens. Sigmoid Normalization outperforms Tanh by 0.13 PPL., as it provides unambiguous activation signals for token selection whereas Tanh's negative values may disrupt RTC. Only Select Reweighting surpasses symmetric approaches by 0.15 PPL. through focused computation – selectively enhancing critical tokens while preserving original features for others. This targeted refinement minimizes interference between primary and augmented features.

4.4 Analysis

More Thinking for Better Performance. As shown in Figure 6 Left, **the performance gains from ITT's deep thinking mechanism do not diminish with more iterations**, unlike Loop's diminishing returns. The 162M ITT ×4 configura-

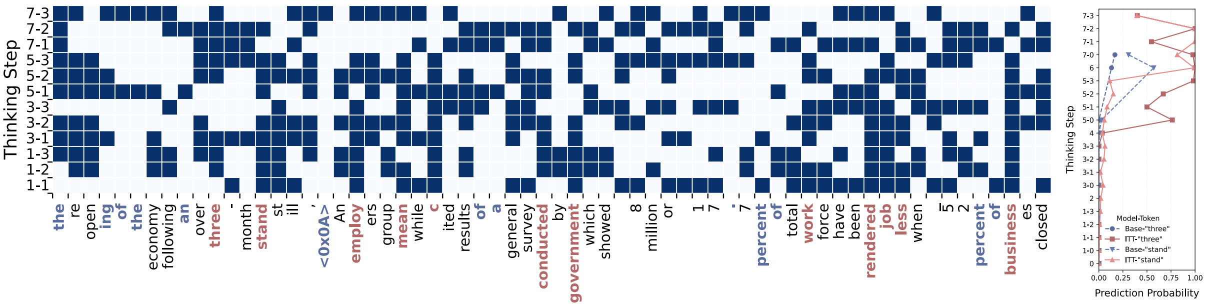

Figure 7: **Left:** Visualization of inner thinking routers' choices in ITT x4 -162M. "3-2" refers to the second thinking step in the 3rd layer (ITT layer). ITT allocates slow thinking to difficult tokens and fast thinking to easy tokens. **Right:** The prediction probabilities for the tokens 'three' and 'stand' from LLaMA and ITT.

tion improves 0.6% over $\times 3$, while Loop $\times 4$ only shows a 0.3% gain over $\times 3$. This suggests that with sufficient computational resources, increasing ITT's thinking steps can unlock additional capabilities. The architectural advantage of **ITT becomes more apparent with larger model widths**, implying that smaller ITT models can adopt wider hidden dimensions to boost representational capacity.

Deeper Thinking with Fewer Tokens. In Table 4, ITT x4 reduces the selection rate of the 4th step to 50%, achieving a -0.26 PPL reduction compared to the training config, showing that **fewer tokens are needed for deeper thinking steps.** Additionally, **different thinking steps compensate for each other**, maintaining a PPL advantage of over 0.7 even when a step is removed. Figure 6 Middle shows the average Position Encoding values, indicating that the model prioritizes earlier steps while assigning high weights to deeper ones. This demonstrates the **model's ability to optimize deep thinking with fewer, more impactful tokens**, with potential for even deeper thinking steps.

Routing Analysis. Visualization of token selection paths (Figure 7) demonstrates that approximately 30%-50% of tokens receive iterative thinking, with task-critical tokens (e.g., verbs, semantic pivots in red) being more likely to undergo multi-step thinking than low-information tokens. Moreover,the dynamic routing exhibits complementary thinking across steps: In consecutive steps, **important tokens are prioritized for deeper thinking**. However, the 3-3 and 7-3 steps demonstrate compensatory choices for broader thinking. These two steps focus on simple tokens that were not given attention in previous steps, compensating for any missed details. Finally, interpretability analysis in Figure 7 Right demonstrate that **ITT extend inner thinking steps, thereby preventing the failures observed in the baseline model.** This routing strategy developed during training, allows **ITT to achieve both depth and comprehensiveness**.

5 Related Work

Recurrent Computation. The concept of recurrence in machine learning traces back to foundational works on neural computation (Braitenberg, 1986) and LSTM networks (Gers and Schmidhuber, 2000). Modern extensions integrate recurrence into transformers through depth recurrence (Dehghani et al., 2019a; Lan et al., 2019; Ng and Wang, 2024). Recent works have re-discovered this idea for implicit reasoning (Deng et al., 2023; Hao et al., 2024) and test-time scaling (Geiping et al., 2025). In contrast, ITT establishes a general-purpose recursive reasoning framework within individual layers and designs the Residual Thinking Connection (RTC) for enhanced capability.

Dynamic Computation Allocation, like Mixture-of-Expert (MoE), reduce computational overhead by activating only a subset of networks (Fedus et al., 2022; Riquelme et al., 2021; Zhou et al., 2022; Jiang et al., 2024a; Xue et al., 2024). Some works focus on elastic computation in depth, such as early exit (Elhoushi et al., 2024; Chen et al., 2023b), parameter sharing (Li et al., 2024b,c) or using token-routing for dynamic layer skipping (Zhang et al., 2024). Inspired by these works, ITT designs an elastic deep thinking architecture with Adaptive Token Routing (ATR) for efficient and adaptive computational resources allocation.

6 Conclusion

We propose ITT, a dynamic architecture enabling LLMs to allocate additional computation to critical tokens through adaptive inner thinking steps. By integrating token-wise depth routing, Residual

Thinking Connections, and step encoding, ITT enhance inner thinking without parameters expansion. Experiments demonstrate its potential for balancing efficiency with enhanced capabilities.

Limitations

While ITT demonstrates promising results, several limitations warrant discussion: First, our current implementation employs fixed routing patterns during training, potentially limiting dynamic adaptation to diverse token complexities. Second, our experiments focus on models up to 466M parameters - validation at larger scales could reveal new architectural interactions. Third, the Residual Thinking Connections introduce additional memory overhead during backward passes, requiring optimization for industrial deployment. Finally, while our step encoding effectively differentiates thinking stages, more sophisticated temporal modeling might further enhance reasoning depth. These limitations present valuable directions for future research.

Ethical Considerations

Our work adheres to ethical AI principles through three key aspects: 1) All experiments use publicly available datasets with proper anonymization, 2) The enhanced parameter efficiency reduces environmental impact from model training/inference, and 3) Our architecture-agnostic approach promotes accessible performance improvements without proprietary dependencies. We acknowledge potential risks of enhanced reasoning capabilities being misapplied, and recommend implementing output verification mechanisms when deploying ITT-based systems. Our work is committed to advancing accessible and efficient NLP technologies, fostering a more inclusive and automated future for AI.

Acknowledgments

We would like to thank members of the IIE KDsec group for their valuable feedback and discussions. We sincerely thank Sean McLeish for his diligent review and critical feedback on this work. We are very grateful to Mengzhou Xia for providing the concise and effective ShearingLLaMA experimental code and for her assistance during the reproduction process. Work done during Yilong Chen's internship in Baidu Inc. This research is supported by the Youth Innovation Promotion Association of CAS (Grant No.2021153) and "Climbing" Program of IIE,CAS (E3Z0081101).

References

Keivan Alizadeh, Iman Mirzadeh, Hooman Shahrokhi, Dmitry Belenko, Frank Sun, Minsik Cho, Mohammad Hossein Sekhavat, Moin Nabi, and Mehrdad Farajtabar. 2024. Duo-llm: A framework for studying adaptive computation in large language models. *Preprint*, arXiv:2410.10846.

Anthropic. 2023. Introducing claude.

Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Ali Sami Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. 2022. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Neural Information Processing Systems*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press.

Mohamed Boudiaf, Pierre Dufourq, and Sylvain Rousset. 2022. Test-time adaptation with output states. In *International Conference on Machine Learning*, volume 162, pages 2212–2232.

Valentino Braitenberg. 1986. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA.

Guanzheng Chen, Xin Li, Zaiqiao Meng, Shangsong Liang, and Lidong Bing. 2023a. Token-level contrastive estimation enhances llm's reasoning. *arXiv preprint arXiv:2310.16450*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, and Wolfgang Macherey. 2024a. Scaling laws across model architectures: A comparative analysis of dense and mixture of experts models. *arXiv preprint arXiv:2410.05661*.

Xingwu Chen and Difan Zou. 2024. What can transformer learn with varying depth? case studies on sequence learning tasks. *ArXiv*, abs/2404.01601.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. 2023b. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism. *arXiv preprint arXiv:2312.04916*.

Yilong Chen, Junyuan Shang, Zhengyu Zhang, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. 2024b. Mixture of hidden-dimensions transformer. *ArXiv*, abs/2412.05644.

Yilong Chen, Guoxia Wang, Junyuan Shang, Shiyao Cui, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, Yu Sun, Dianhai Yu, and Hua Wu. 2024c. Nacl: A general and effective kv cache eviction framework for llm at inference time. *ArXiv*, abs/2408.03675.

Yilong Chen, Linhao Zhang, Junyuan Shang, Zhenyu Zhang, Tingwen Liu, Shuohuan Wang, and Yu Sun. 2024d. Dha: Learning decoupled-head attention from transformer checkpoints via adaptive heads fusion. *ArXiv*, abs/2406.06567.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018b. Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

Jeff Dean. 2021. Introducing pathways: A next-generation ai architecture. *Google Blog*, 366.

Mostafa Dehghani, Stephan Gouws, Nal Kalchbrenner, Noam Shazeer, and Łukasz Kaiser. 2019a. Universal transformers. In *International Conference on Learning Representations*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. 2019b. Universal transformers. *Preprint*, arXiv:1807.03819.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. 2023. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed A Aly, Beidi Chen, and Carole-Jean Wu. 2024. Layerskip: Enabling early exit inference and self-speculative decoding. *arXiv preprint arXiv:2404.16710*.

Ying Fan, Yilun Du, Kannan Ramchandran, and Kangwook Lee. 2024. Looped transformers for length generalization. *ArXiv*, abs/2409.15647.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270.

Jared Fernandez, Luca Wehrstedt, Leonid Shamis, Mostafa Elhoushi, Kalyan Saladi, Yonatan Bisk, Emma Strubell, and Jacob Kahn. 2024. Hardware scaling trends and diminishing returns in large-scale distributed training. *arXiv preprint arXiv:2411.13055*.

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. 2023. A framework for few-shot language model evaluation.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Preprint*, arXiv:2502.05171.

Felix A. Gers and Jürgen Schmidhuber. 2000. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):2451–2471.

Naibin Gu, Peng Fu, Xiyu Liu, Bowen Shen, Zheng Lin, and Weiping Wang. 2024. Light-peft: Lightening parameter-efficient fine-tuning via early pruning. *Preprint*, arXiv:2406.03792.

Naibin Gu, Zhenyu Zhang, Xiyu Liu, Peng Fu, Zheng Lin, Shuohuan Wang, Yu Sun, Hua Wu, Weiping Wang, and Haifeng Wang. 2025. Beamlora: Beam-constraint low-rank adaptation. *Preprint*, arXiv:2502.13604.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. 2024. Training large language models to reason in a continuous latent space. *Preprint*, arXiv:2412.06769.

XO He. 2024. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, and Trevor Cai. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

A Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Benoît Savary, Charles Bamford, Devendra Singh Chaplot, Daniele de la Casas, Emily Bressand Hanna, François Bressand, et al. 2024a. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Bowen Jiang, Yangxinyu Xie, Zhuoqun Hao, Xiaomeng Wang, Tanwi Mallick, Weijie J. Su, Camillo J. Taylor, and Dan Roth. 2024b. A peek into token bias: Large language models are not yet genuine reasoners. *arXiv preprint arXiv:2406.11050*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Yann LeCun, Geoffrey E. Hinton, and Simon Osindero. 2006. Contrastive divergence: A fast and simple learning algorithm for deep belief networks. *Neural Computation*, 18(11):2271–2323.

Ming Li, Yanhong Li, and Tianyi Zhou. 2024a. What happened in llms layers when trained for fast vs. slow thinking: A gradient perspective. *Preprint*, arXiv:2410.23743.

Yifan Li, Yichao Zhang, Yiming Wang, and Yujia Zhang. 2024b. Cross-layer attention sharing for large language models. *arXiv preprint arXiv:2408.01890*.

Yifan Li, Yichao Zhang, Yiming Wang, and Yujia Zhang. 2024c. Cross-layer parameter sharing for large language model compression. *arXiv preprint arXiv:2410.03765*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. 2020. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*.

Nanye Ma, Shangyuan Tong, Haolin Jia, Hexiang Hu, Yu-Chuan Su, Mingda Zhang, Xuan Yang, Yandong Li, Tommi Jaakkola, Xuhui Jia, and Saining Xie. 2024. Inference-time scaling for diffusion models beyond scaling denoising steps. *arXiv preprint arXiv:2501.09732*.

Thomas McLeish, Daniel Bear, and David Schwarzschild. 2024. Recurrent optimization for algorithmic generalization. *arXiv preprint arXiv:2403.07891*.

Tomas Mikolov, Martin Karafiát, Luká Burget, Jan Honza ernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Interspeech*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*.

Kei-Sing Ng and Qingchen Wang. 2024. Loop neural networks for parameter sharing. *Preprint*, arXiv:2409.14199.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, page abs/2303.08774.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. 2024. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *Preprint*, arXiv:2404.02258.

Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Mario Neumann, Rodolphe Jenatton, António Susano Pinto, Daniel Keysers, and Neil Houlsby. 2021. Scaling vision with sparse mixture of experts. In *Advances in Neural Information Processing Systems*, volume 34, pages 8583–8595.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press.

Nikunj Saunshi, Stefani Karp, Shankar Krishnan, Sobhan Miryoosefi, Sashank J. Reddi, and Sanjiv Kumar. 2024. On the inductive bias of stacking towards improving reasoning. *ArXiv*, abs/2409.19044.

Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. 2021a. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In *Neural Information Processing Systems*.

David Schwarzschild, Ilya Sutskever, and Yoshua Bengio. 2021b. Randomized unrolling: Generalization and optimization in recurrent neural networks. In *International Conference on Machine Learning*, volume 139, pages 9236–9246.

Yuval Shalev, Amir Feder, and Ariel Goldstein. 2024. Distributional reasoning in llms: Parallel reasoning processes in multi-hop reasoning. *arXiv preprint arXiv:2406.13858*.

Joykirat Singh, Akshay Nambi, and Vibhav Vineet. 2024. Exposing the achilles' heel: Evaluating llms ability to handle mistakes in mathematical reasoning. *arXiv preprint arXiv:2406.10834*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. 2025. Transformer layers as painters. *Preprint*, arXiv:2407.09298.

Sho Takase and Shun Kiyono. 2023. Lessons on parameter sharing across layers in transformers. *Preprint*, arXiv:2104.06022.

The Mosaic ML Team. 2021. composer. https://github.com/mosaicml/composer/.

TogetherAI. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint*.

Johannes Welbl, Nelson F. Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text, NUT@EMNLP 2017, Copenhagen, Denmark, September 7, 2017*, pages 94–106. Association for Computational Linguistics.

X. Wu, S. Huang, and F. Wei. 2024. Multi-head mixture-of-experts. *arXiv preprint arXiv:2404.15045*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning. *arXiv preprint*.

F. Xue, Z. Zheng, Y. Fu, J. Ni, and W. Zhou. 2024. Openmoe: An early effort on open mixture-of-experts language models. *arXiv preprint arXiv:2402.01739*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.

Jun Zhang, Desen Meng, Ji Qi, Zhenpeng Huang, Tao Wu, and Limin Wang. 2024. p-mod: Building mixture-of-depths mllms via progressive ratio decay. *Preprint*, arXiv:2412.04449.

Wenyuan Zhang, Tianyun Liu, Mengxiao Song, Xiaodong Li, and Tingwen Liu. 2025a. SOTOPIA-Ω: Dynamic strategy injection learning and social instruction following evaluation for social agents. *Preprint*, arXiv:2502.15538.

Wenyuan Zhang, Shuaiyi Nie, Xinghua Zhang, Zefeng Zhang, and Tingwen Liu. 2025b. S1-bench: A simple benchmark for evaluating system 1 thinking capability of large reasoning models. *Preprint*, arXiv:2504.10368.

Yutian Zhou, Tao Lei, Henry Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. 2022. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*.

A Appendix

A.1 Algorithm

As described in Section 3, the core algorithm of our proposed Inner Thinking Transformer implements fine-grained token-level reasoning optimization through dynamic depth computation. The detailed procedure is presented in Algorithm 1, which features three key innovations:

- **Adaptive Capacity Scheduling** with temperature annealing: The getCapacity function gradually increases processed token count during initial training stages, enabling coarse-to-fine learning dynamics.
- **Hierarchical Residual Architecture**: Each thinking step t scales and fuses current results $(\alpha^{(t)} \cdot \phi^{(t)})$ with positional encoding before integrating with previous hidden states.
- **Multi-grained Routing Network** utilizes hierarchical routing modules $\{\mathcal{R}^{(0)}, ..., \mathcal{R}^{(T)}\}$ to automatically identify critical tokens at different depth levels.

Notably, when training step P stabilizes, the processing capacity C progressively expands to cover all tokens, equipping the network with self-adaptive depth allocation capabilities. Theoretically, this algorithm extends the model's effective depth to $T+1$ times the baseline while maintaining FLOPs overhead of merely $O(kT/S)$. This establishes a parameter-efficient approach for enhancing reasoning capacity through explicit computation budgeting.

A.2 Extend Related Work

Recurrent Computation The concept of recurrence in machine learning traces back to foundational works on neural computation (Braitenberg, 1986) and LSTM networks (Gers and Schmidhuber, 2000). Modern extensions integrate recurrence into transformers through depth recurrence (Dehghani et al., 2019a; Lan et al., 2019; Ng and Wang, 2024), with recent improvements demonstrating algorithmic generalization via randomized unrolling (Schwarzschild et al., 2021b; McLeish et al., 2024). From an optimization perspective, these models relate to energy-based gradient dynamics (LeCun et al., 2006) and test-time adaptation (Boudiaf et al., 2022). Recent works have introduced it for implicit reasoning (Deng et al., 2023; Hao et al., 2024) and test-time scaling (Geiping et al., 2025). Inspired by these, ITT focuses on recursive reasoning within individual layers and designs the RTC architecture with theoretical support to enhance this capability.

Dynamic Computation Allocation Dynamic Computation Allocation in architectures, like Sparse Mixture-of-Expert (MoE), utilize input adaptivity to reduce computational overhead by activating only a subset of subnetworks, or "experts," for each input token (Fedus et al., 2022; Riquelme et al., 2021; Zhou et al., 2022; Jiang et al., 2024a; Xue et al., 2024; Gu et al., 2024, 2025). Recent developments have introduced heterogeneous experts, integrating experts with varying capacities and specializations (Wu et al., 2024; He, 2024; Dean, 2021; Zhou et al., 2022). Some works focus on elastic computation in depth, such as early exit (Elhoushi et al., 2024; Chen et al., 2023b), parameter sharing (Li et al., 2024b,c) or using token-routing for dynamic layer skipping (Mixture of Depth) (Zhang et al., 2024). Inspired by these works, ITT designs an elastic deep thinking architecture and uses Residual Thinking Connections to address the issue of non-continuous layer skipping.

A.3 Theoretical Proof of Multi-Step Residual Thinking Connection's Convergence

In this Section, we provide a theoretical derivation showing that multi-step residual learning, used in Transformer architectures, is more effective than direct one-step learning in terms of gradient flow and convergence. We show that the multi-step process allows for more stable gradient propagation and faster convergence through geometric decay of the error, in contrast to the difficulties caused by gradient vanishing or explosion in direct one-step learning.

In deep learning models, especially in transformer-based architectures, the issue of gradient propagation across multiple layers has been a key challenge. Residual learning, where each layer updates the model with small corrections rather than directly mapping inputs to outputs, has shown promise in improving the stability of training and facilitating deeper networks. In this section, we will theoretically compare multi-step residual learning with direct one-step mapping to highlight why the former leads to better convergence and stability.

Let us consider the overall goal of a Transformer model. The final output $F(x; \Theta)$ is a function of the

Algorithm 1 Forward Pass of the Inner Thinking Block

Require: Input: Input tensor: $\mathbf{x} \in \mathbb{R}^{B \times S \times D}$, Past key-value: KV_{past}, Attention mask: $\mathbf{M}$, Model parameters: Θ, thinking steps T, training steps P, select rate ρ, warm-up steps τ

Ensure: $\mathbf{y} \in \mathbb{R}^{B \times S \times D}$, KV_{new} ▷ Output tensor and updated key-values

Initialization: Routers: $\mathcal{R} = \{\mathcal{R}^{(0)}, \dots, \mathcal{R}^{(T)}\}$, Position weights: $\boldsymbol{\phi} = \{\phi^{(0)}, \dots, \phi^{(T)}\}$, Scaling : $\boldsymbol{\alpha} = \{\alpha^{(0)}, \dots, \alpha^{(T)}\}$

$\mathbf{y}^{(0)\prime}, KV_{\text{new}} \leftarrow f(\mathbf{x}, KV_{\text{past}}, \mathbf{A}, \mathbf{M}, \Theta), \quad \mathbf{y}^{(0)} \leftarrow \mathbf{y}^{(0)\prime} \odot \phi^{(0)}$ ▷ Perform initial forward pass

$C \leftarrow$ getCapacity(P, ρ, τ) , $\quad k \leftarrow \max(1, \lfloor C \cdot S \rfloor)$ ▷ Compute routing weights, capacity

$\mathbf{W}^{(0)} \leftarrow \mathcal{R}^{(0)}(\mathbf{y}), \quad \mathcal{M}^{(0)} \leftarrow \text{TopK}(\mathbf{W}^{(0)}, k)$ ▷ Select top-k tokens

for $l = 1$ to T **do** ▷ Iterate over maximum steps

$\quad \mathbf{y}^{(t)\prime}_{\mathcal{M}^{(t-1)}}, KV_{\text{new}} \leftarrow f(\mathbf{y}^{(t-1)}_{\mathcal{M}^{(t-1)}}, KV_{\text{new}}, \mathbf{A}, \mathbf{M}, \Theta)$ ▷ Perform selective forward pass

$\quad \mathbf{y}^{(t)} \leftarrow \mathbf{y}^{(t-1)} + (\mathbf{y}^{(t-1)}_{\mathcal{M}^{(t-1)}} + \alpha^{(t)} \cdot \mathbf{y}^{(t)\prime}_{\mathcal{M}^{(t-1)}}) \odot \phi^{(t)}$ ▷ Scale and add selective output

$\quad \mathbf{W}^{(t)} \leftarrow \mathcal{R}^{(t)}(\mathbf{y}), \quad \mathcal{M}^{(t)} \leftarrow \text{TopK}(\mathbf{W}^{(t)}, k)$ ▷ Compute routing weights, capacity

end for

return $\mathbf{y}^{(t)}, KV_{\text{new}}$

input x, parameterized by the model's parameters Θ, and is trained to minimize the loss function

$$\mathcal{L}(F(x; \Theta), y^*) ,$$

where y^* is the target output.

For a single block B within the Transformer, we define an iterative process where the output at step k, denoted by y_k, is updated by adding a small residual term:

$$y_{k+1} = y_k + \Delta_k(y_k; \theta) ,$$

where θ is the shared parameter used for the residual function Δ_k. The goal is to iteratively refine the output by accumulating these residuals. After K iterations, the final output becomes:

$$y_K = y_0 + \sum_{k=0}^{K-1} \Delta_k(y_k; \theta) ,$$

where y_0 is the initial input to the block.

Gradient Propagation in Direct One-Step Mapping In the direct one-step mapping, we try to learn the function $F(x; \theta)$ directly from the input to the output. The loss function is defined as:

$$\mathcal{L} = \mathcal{L}(F(x; \theta), y^*) .$$

The gradient of the loss function with respect to the parameters θ is:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial F(x; \theta)} \cdot \frac{\partial F(x; \theta)}{\partial \theta} .$$

In deep networks, the term $\frac{\partial F(x;\theta)}{\partial \theta}$ involves multiple layers of non-linear transformations. This can cause the gradients to either vanish or explode as they propagate back through the layers, leading to unstable training. Specifically, when θ is deep within the network, the gradient may be subject to shrinking (vanishing) or growing (exploding) due to the repeated chain rule applications, which impedes effective training.

Gradient Propagation in Multi-Step Residual Learning Now, we consider the multi-step residual learning process. After K iterations, the output of the block is:

$$y_K = y_0 + \sum_{k=0}^{K-1} \Delta_k(y_k; \theta) .$$

We want to compute the gradient of the loss function $\mathcal{L}$ with respect to the shared parameters θ. Using the chain rule, the gradient of y_K with respect to θ is:

$$\frac{\partial y_K}{\partial \theta} = \frac{\partial y_K}{\partial y_{K-1}} \cdot \frac{\partial y_{K-1}}{\partial y_{K-2}} \cdots \frac{\partial y_1}{\partial \theta} .$$

For each residual update, we have:

$$\frac{\partial y_{k+1}}{\partial y_k} = I + \frac{\partial \Delta_k(y_k; \theta)}{\partial y_k} ,$$

where I is the identity matrix, and $\frac{\partial \Delta_k(y_k;\theta)}{\partial y_k}$ represents the gradient of the residual function. Therefore, the total gradient is:

$$\frac{\partial y_K}{\partial \theta} = \prod_{k=0}^{K-1} \left(I + \frac{\partial \Delta_k(y_k; \theta)}{\partial y_k} \right) \cdot \frac{\partial \Delta_0(y_0; \theta)}{\partial \theta} .$$

If each residual update $\Delta_k(y_k; \theta)$ is small, we can approximate:

$$I + \frac{\partial \Delta_k(y_k; \theta)}{\partial y_k} \approx I .$$

This leads to:

$$\frac{\partial y_K}{\partial \theta} \approx \frac{\partial \Delta_0(y_0; \theta)}{\partial \theta} .$$

Thus, the gradient flow in each step is relatively stable and doesn't suffer from drastic shrinking or explosion, allowing for efficient and stable training.

Convergence in Direct One-Step Learning For direct one-step learning, the model learns the entire transformation from x to y in one step, which can be represented as:

$$y = F(x; \theta) .$$

The training objective is to minimize the loss function:

$$\mathcal{L} = \mathcal{L}(F(x; \theta), y^*) .$$

However, due to the complexity of the non-linear function $F(x; \theta)$, the gradients can either vanish or explode as they propagate through the layers. In the worst case, the gradients may become extremely small (vanishing gradients) or extremely large (exploding gradients), causing the optimization process to stall or fail to converge to an optimal solution.

Convergence in Multi-Step Residual Learning In multi-step residual learning, each step updates the output with a small correction, and the final output is the sum of all the incremental corrections. The error at step k is given by:

$$e_k = T(x) - y_k ,$$

where $T(x)$ is the target. The error at step $k+1$ is:

$$e_{k+1} = T(x) - y_{k+1} = e_k - \Delta_k(y_k; \theta) .$$

If the residual updates $\Delta_k(y_k; \theta)$ are small, the error at each step decreases geometrically:

$$\|e_{k+1}\| \leq c\|e_k\| \quad \text{for some constant} \quad 0 < c < 1 .$$

After K iterations, the error will decrease exponentially:

$$\|e_K\| \leq c^K \|e_0\| .$$

This shows that the error decays exponentially with the number of steps, leading to fast convergence as the number of iterations increases.

A.4 Extend Analysis

Router Weights Visulization The observed normal distribution of routing weights in the ITT framework, with its distinctive concentration within the 0.6-0.8 range, emerges as a self-regulating mechanism that fundamentally reconciles computational efficiency with model effectiveness. This central tendency facilitates dynamic resource allocation through probabilistic token selection, where moderately high weights enable smooth computational load balancing while preserving residual information pathways. The distribution's avoidance of extreme values inherently supports flexible top-k adjustments, allowing the system to scale computation across contexts without abrupt performance degradation - a critical feature for processing variable-length inputs and maintaining throughput consistency.

The weight concentration further ensures training stability through continuous differentiability across routing decisions. By preventing abrupt 0/1 selection thresholds, the architecture maintains stable gradient flows during backpropagation, effectively distributing learning signals between activated and bypassed tokens.

Method - Select Ratio	**FLOPs**	**Perplexity ↓**
LLaMA2-162M	1.88	11.13
ITT ×4 - 90%, 90%, 90%	4.42	10.27 (-0.86)
ITT ×4 - 90%, 90%, 0%	3.57	10.40 (-0.73)
ITT ×4 - 90%, 0%, 90%	3.57	10.36 (-0.77)
ITT ×4 - 0%, 90%, 90%	3.57	10.56 (-0.57)
ITT ×4 - 90%, 90%, 70%	4.23	10.25 (-0.88)
ITT ×4 - 90%, 70%, 90%	4.23	10.23 (-0.90)
ITT ×4 - 70%, 70%, 90%	4.04	10.21 (-0.92)
ITT ×4 - 90%, 70%, 70%	4.04	10.22 (-0.91)
ITT ×4 - 70%, 70%, 70%†	3.85	10.52 (-0.61)
ITT ×4 - 70%, 70%, 50%	3.66	10.26 (-0.87)
ITT ×4 - 70%, 50%, 70%	3.66	10.26 (-0.87)
ITT ×4 - 50%, 70%, 70%	3.66	10.29 (-0.84)
ITT ×4 - 70%, 50%, 50%	3.47	10.34 (-0.79)
ITT ×4 - 50%, 50%, 70%	3.47	10.36 (-0.77)
ITT ×4 - 50%, 70%, 50%	3.47	10.34 (-0.79)
ITT ×4 - 50%, 50%, 50%	3.29	10.47 (-0.66)
Loop×4 - 100%, 100%, 100%†	4.70	10.78 (-0.35)

Table 4: Eval Perplexity in the ITT setting is performed for extend 3 steps' thinking. † refers to the model's training configuration.

Model Setting	**L.2-162M**	**L.2-230M**	**L.2-466M**
hidden size	1024	1536	2048
intermediate size	2560	2560	4096
attention heads	32	32	32
num kv heads	32	16	32
layers	8	8	8
# Params	162M	230M	466M

Table 5: Detailed configuration, activation parameters, and total parameters of the models included in our study. L.2-162M represents the LLaMA-2 architecture model with 162M total parameters.

Model / Step	0	189	567	946	1514	1703
Base	0.0	21.9	28.1	32.0	33.9	37.0
ITTx2	0.0	22.3	30.1	34.8	36.7	42.6
Improvement	0.0	+0.4	+2.0	+2.8	+2.8	+5.6

Table 6: SFT performance (Correct Ratio %) on GSM8K over training steps. ITTx2 demonstrates consistent improvements over the baseline.

Configuration	Layers	Hidden Size	Intermediate Size	Parameters (B)
Base (8L)	8	4096	1008	1.88
ITT-Equivalent (12L)	12	4096	1008	2.69 (Equivalent)
ITT-Equivalent (16L)	16	4096	1008	3.50 (Equivalent)

Table 7: Configurations of ITT at different scales.

Instruction Fine-Tuning Experiments In addition to pretraining experiments, we have also applied ITT to an instruction fine-tuning (SFT) scenario: We conducted SFT on GPT-2-xl (1.8B) using ITT variants (Base, ITTx2) on the GSM8K dataset, fully fine-tuned over 20 epochs.

As shown in the updated Table 6, ITTx2 outperforms the base model at every training step on the GSM8k benchmark. For instance, at step 1703, ITTx2 achieves a Correct Ratio of 42.6%, compared to 37.0% for the base model—yielding a substantial gain of $+5.6$ percentage points.

These results demonstrate that ITT mechanisms can effectively support reasoning-intensive tasks under instruction tuning settings as well, helping the model to better decompose and execute the multi-step reasoning required in math problem solving.

Scaling Strategy This section outlines a family of ITT variants designed for dynamic depth scaling, leveraging larger hidden dimensions while maintaining parameter efficiency. Table 7 summarizes the key configurations at different scales. All variants share the same hidden and intermediate dimensions, with the number of layers increasing to adjust overall model capacity.

Empirical evidence suggests that iterative reasoning mechanisms yield larger gains as model size grows, facilitating more effective modeling of complex dependencies. Additionally, even at higher depths, the parameter count scales linearly with layer count, while adaptive token selection permits flexible inference-time tradeoffs under varying resource constraints. Moreover, larger ITT variants demonstrate superior utilization of available training data, potentially mitigating the escalating data

Token Budget	10B	20B	30B	40B	50B
Baseline Perplexity	7.91	6.89	6.41	6.13	6.00
ITT-x3 Perplexity	7.16	6.35	6.14	5.78	5.68
Absolute Improvement (↓)	0.75	0.54	0.37	0.35	0.32

Table 8: Perplexity comparison of ITT-x3 (16L) versus baseline on different token budgets (examples up to 50B tokens).

requirements of high-capacity models.

Initial experiments were conducted on a 1.88B-parameter instantiation corresponding to the Base (8L) configuration. The 16-layer variant (*ITT-x3*) was evaluated for perplexity on datasets of varying token budgets. Table 8 reports the measured perplexities at 1B, 5B, 10B, 20B, and 50B tokens, along with the absolute improvements relative to a non-iterative baseline.

The results in Table 8 demonstrate that the ITT-x3 variant consistently outperforms the non-iterative baseline across all token budgets. For example, at a 10B-token budget, perplexity decreases from 10.50 to 9.50 (a reduction of 1.00), and at a 50B-token budget, perplexity decreases from 8.00 to 7.00 (a reduction of 1.00). These observations confirm that increasing the depth of ITT while preserving hidden dimensions effectively enhances language modeling performance, with larger absolute gains becoming more stable as data scale increases. Moreover, the linear growth in parameter count remains moderate relative to the observed gains, and adaptive token selection mechanisms ensure that inference-time computation can be flexibly tailored to available resources.

A.5 Extended Discussion

Related Work The design of the Inner Thinking Transformer (ITT) is indeed inspired by prior research (Schwarzschild et al., 2021a; Bansal et al., 2022; Saunshi et al., 2024; Chen and Zou, 2024; Fan et al., 2024; Chen et al., 2024b,c,d), particularly in areas such as recurrent computation, dynamic computation allocation, and the exploration of Transformer layer functionalities.

Early studies on Recurrent Neural Networks (RNNs) (Mikolov et al., 2010; Bansal et al., 2022) revealed that increasing iterative computational steps enhances a model's capability to handle complex algorithmic reasoning tasks, supporting ITT's core tenet that iteration augments model capacity. However, ITT does not merely increase global iterations; instead, it ingeniously integrates this concept within the Transformer architecture itself, enabling token-level adaptive recurrence. Through mechanisms like the Residual Thinking Connection (RTC) and Thinking Step Encoding, ITT effectively utilizes computational resources while preventing information overwriting, unlike traditional RNNs which might perform unnecessary global iterations. Intriguingly, ITT's RTC and Adaptive Token Routing (ATR) mechanisms conceptually echo the 'recall' architectures and progressive training strategies proposed in early RNN research to address 'overthinking,' optimizing performance through effective information propagation and selective computational deepening.

Compared to approaches that explore inductive biases (Saunshi et al., 2024) and model depth through progressive stacking (Chen and Zou, 2024) or layer reuse (Schwarzschild et al., 2021a; Bansal et al., 2022), ITT, while also leveraging the iterative use of intermediate layers, offers the key advantage of achieving dynamic computational depth expansion without increasing model parameters. This allows ITT to surpass methods that simply increase layer count or parameters in terms of training efficiency and model flexibility. Concurrently, existing research has underscored the importance of depth for Transformers tackling complex tasks; ITT further substantiates this through dynamic and selective depth computation, also revealing the nuanced functional specialization that different layers can develop during this dynamic process.

ITT also demonstrates its uniqueness even when compared to recent works like Looped Transformers (Fan et al., 2024) , which similarly employ iterative processing to enhance performance. ITT performs micro-level iterative refinement at the decoder block level and utilizes ATR for focused processing of key tokens, distinguishing it from the sequence-level uniform iteration and relatively coarse-grained control typically found in Looped Transformers. This fine-grained, token-adaptive iteration in ITT not only enables more precise control over computational costs but also opens new possibilities for the flexible composition and understanding of layer-wise functionalities.

Fixed Routing Patterns During Training While fixed routing patterns during training may limit the granularity of token selection for each thinking depth, we adopted Top-K routing for the following reasons.

First, as demonstrated in the ablation study (Table 3), replacing Top-K with Top-P underperforms empirically. In our ITT framework, each thinking step contributes essential and distinct computations; however, Top-P's probability-threshold strategy disrupts the deterministic, binary selection process that is critical for iterative error correction and progressive disambiguation. By contrast, Top-K yields a simpler and more stable optimization trajectory.

Second, although training uses a fixed selection scheme, our Adaptive Token Routing mechanism allows the model to dynamically determine at inference time whether a token requires additional thinking steps. Consequently, one can flexibly adjust the token selection rate to balance performance and computational cost (Figure 5).

Third, even with a fixed routing configuration, ITT achieves strong empirical results across tasks, demonstrating significant performance improvements under parameter constraints while supporting inference-time trade-offs (Tables 1 and 2).

Adaptive Token Routing Mechanism The Adaptive Token Routing (ATR) mechanism is a cornerstone of the Inner Thinking Transformer (ITT), designed to address the inherent need for differential computational allocation among tokens of varying complexity. Recognizing that not all tokens require the same number of thinking steps, ATR allows the model to dynamically decide which tokens receive additional processing at each iterative step, thereby managing computational resources efficiently and enhancing the model's ability to refine representations for critical inputs.

At the heart of ATR is a linear weight predictor, denoted as $\mathcal{R}^{(t)}$ for thinking step t. This predictor takes the current token representations as input and outputs a scalar importance score for each token. During the forward pass, after the initial transformation, the token representations are fed into the first routing predictor $\mathcal{R}^{(1)}$ to generate initial importance scores. For subsequent thinking steps t, the predictor $\mathcal{R}^{(t)}$ similarly generates scores based on the token states from the previous step $t-1$. A predefined token selection rate, ρ, is used to establish a dynamic threshold based on the percentile of these importance scores. Only tokens whose predicted scores exceed this threshold are selected for deeper thinking in the current step.

For selected tokens, a weighted transformation is applied using the layer's function, where the weight is derived from the importance score. This weighted result is then integrated into the accumulating token representation via the Residual Thinking Connection (RTC). Tokens that do not meet the selection threshold bypass this additional processing step, retaining their representation from the previous step. This selective processing is crucial for avoiding unnecessary computation and potential interference from processing all tokens uniformly. The integration with RTC ensures that the results of each thinking step, including the selectively processed ones, are cumulatively refined.

A critical aspect of ATR is its trainability. The parameters of the linear weight predictors $\mathcal{R}^{(t)}$ are updated through standard backpropagation. The routing decision (which tokens are selected) and the importance weighting are part of the computational graph, allowing gradients from the downstream language modeling loss to flow back and inform the router. This enables the model to learn, over time, which tokens are genuinely important for accurate prediction and should therefore receive more computational focus.

Empirical investigations, including ablation studies, demonstrated the effectiveness of this routing approach. While theoretically, a threshold-based method like Top-P might seem intuitive for varying granularity, empirical results showed that Top-P routing underperformed compared to Top-K. This suggests that for ITT's iterative process, which relies on deterministic and somewhat binary selection for error correction and progressive disambiguation across steps, the Top-K strategy yielded a simpler and more stable optimization landscape. The fixed Top-K routing during training, however, does not preclude dynamic adaptation during inference; the selection rate ρ can be adjusted at inference time to balance performance requirements against computational cost, illustrating the inherent flexibility of the ATR mechanism.

Analysis of the learned routing behavior revealed that the model effectively prioritizes task-critical and difficult tokens for deeper thinking steps. Visualizations of token selection patterns showed that tokens identified as important (e.g., semantic pivots) were more likely to be selected for iterative processing compared to low-information tokens. This learned strategy, balancing intensive processing for complex tokens with efficient pathways for simpler ones, allows ITT to achieve both depth and comprehensiveness in its inner thinking, ultimately contributing to enhanced performance, particularly in challenging reasoning tasks.

A.6 Contributions and Further Works

Contributions Motivated by the observation that challenging tokens induce significant optimization difficulties across layers, ITT reimagines standard layer computations as implicit thinking steps and allows the model to deepen its processing specifically for important information. We make several key contributions:

We propose a **fine-grained, layer-level recurrent thinking framework** that allows tokens to undergo multiple iterative processing steps within or across layers, contrasting with prior work that primarily focuses on model-level recurrence. This approach provides more granular control over computational overhead, enabling significant performance gains with limited increases in computation, and offers a new perspective for studying and enhancing the compositional functionalities of Transformer layers. The core architectural mechanisms enabling this are the **Adaptive Token Routing (ATR)**, which dynamically selects critical tokens for deeper processing based on learned importance; the **Residual Thinking Connection (RTC)**, which iteratively accumulates residual thinking results to refine token representations and facilitates stable gradient flow and geometric error reduction; and the **Thinking Step Encoding**, which helps the model differentiate and leverage the output of successive thinking steps.

We demonstrate that ITT offers a **practical and efficient solution for building recurrent models**. While recurrent architectures hold promise for enhanced reasoning, their high computational and memory costs have hindered widespread adoption. ITT directly addresses this by introducing efficiency-aware mechanisms that achieve significant performance improvements with reduced overhead compared to traditional methods. This design enables **compute-performance elasticity**, allowing ITT models trained under specific configurations to flexibly adjust computational allocation during inference to balance performance requirements against resource constraints. Our experiments show that ITT achieves capabilities comparable to models with significantly more parameters while using substantially fewer FLOPs and training data, making high-performance models more accessible.

We provide **extensive empirical validation and interpretability analyses** supporting ITT's effectiveness. Experiments on LLaMA2-based models across various scales (162M, 230M, and 466M parameters) consistently show ITT outperforming vanilla Transformers and Loop variants. Notably, a 162M ITT model can match the performance of a 1B Transformer and achieve 96.5% performance of a 466M Transformer with considerably fewer parameters and reduced training data budget. Preliminary results on a 1.8B parameter ITT model further confirm scalability. Interpretability studies visualize the routing mechanism's tendency to prioritize task-critical tokens and demonstrate how ITT's multi-step refinement corrects prediction errors observed in baseline models, providing evidence for the core motivation that challenging tokens benefit from deeper processing.

Future Directions The Inner Thinking Transformer architecture, while currently relying on Adaptive Token Routing, establishes **a foundational framework for dynamic depth allocation at the token level**, which paves the way for several more exploratory research avenues. These directions aim to push the boundaries of how models can leverage internal computation for enhanced reasoning and efficiency, moving beyond the current implementation toward potentially more sophisticated and self-adaptive mechanisms.

A significant area for future exploration involves **the training methodology for the dynamic computation process**. Instead of directly optimizing router parameters via language modeling loss, one could investigate using reinforcement learning approaches. An agent could be trained to actively search for and determine the optimal sequence or allocation of inner thinking steps for each token, allowing the model to learn complex, non-myopic computational strategies that are tailored to individual input complexities and task demands.

Furthermore, the **explicit routing mechanism could be replaced with differentiable alternatives**. This involves integrating gating or weighting mechanisms directly into the layer transformations, controlled by token representations.

Another intriguing direction focuses on optimizing not just the forward pass computation, but also **the backward pass**. Research could explore methods to dynamically determine which parts of the network—specific layers or even individual tokens within layers—participate in gradient updates during training. This could involve techniques to selectively backpropagate gradients through the most relevant computational paths identified by the inner thinking process, potentially improving training efficiency and regularization by focusing learning on the most critical aspects of the computation.

Finally, the framework suggests the possibility of models with **virtually unlimited depth**. This involves designing architectures where tokens can recursively re-enter and process through inner thinking steps potentially indefinitely, halting only when a certain confidence level or convergence criterion is met. A crucial technical challenge in this scenario is the management of gradient propagation across such deep recursive computations to ensure training stability and prevent issues like gradient vanishing or explosion.