# IIET: Efficient Numerical Transformer via Implicit Iterative Euler Method

**Anonymous ACL submission**

## Abstract

High-order numerical methods enhance Transformer performance in tasks like NLP and CV, but introduce a performance-efficiency trade-off due to increased computational overhead. Our analysis reveals that conventional efficiency techniques, such as distillation, can be detrimental to the performance of these models, exemplified by PCformer. To explore more optimizable ODE-based Transformer architectures, we propose the **I**terative **I**mplicit **E**uler **T**ransformer **(IIET)**, which simplifies high-order methods using an iterative implicit Euler approach. This simplification not only leads to superior performance but also facilitates model compression compared to PCformer. To enhance inference efficiency, we introduce **I**teration **I**nfluence-**A**ware **D**istillation **(IIAD)**. Through a flexible threshold, IIAD allows users to effectively balance the performance-efficiency trade-off. On lm-evaluation-harness, IIET boosts average accuracy by 2.65% over vanilla Transformers and 0.8% over PCformer. Its efficient variant, E-IIET, significantly cuts inference overhead by 55% while retaining 99.4% of the original task accuracy. Moreover, the most efficient IIET variant achieves an average performance gain exceeding 1.6% over vanilla Transformer with comparable speed.

## 1 Introduction

The integration of advanced numerical Ordinary Differential Equation (ODE) solvers into Transformer architectures (Vaswani, 2017) has spurred significant progress in natural language processing (NLP) (Li et al., 2022, 2024; Tong et al., 2025) and image synthesis (Ho et al., 2020; Lu et al., 2022a,b; Zheng et al., 2024). Leveraging high-order methods, particularly Predictor-Corrector (PC) schemes, within Transformer residual connections has demonstrated the capacity to enhance model learning without increasing parameter counts, offering a pathway to both performance and parameter efficiency (Li et al., 2022, 2024).

However, the promise of high-order PCformer (Li et al., 2024) is often constrained by deployment inefficiencies. The inherent linear dependency in nested computations across layers during inference poses critical inference latency. A straightforward approach to mitigating this deployment bottleneck is Knowledge Distillation (Hinton, 2015; Kim and Rush, 2016). However, our preliminary experiments demonstrate that the inherent architectural discrepancy between the predictor and corrector within PCformer impedes effective knowledge transfer via distillation. Our empirical investigations reveal an obvious 54% loss in performance advantage for distilled student models, even for those initialized with PCformer parameters.

Confronted with these deployment bottlenecks, we pivot towards architectural innovations grounded in numerical method principles. A naive yet seemingly logical initial approach might be to pursue uniformity in numerical methods between predictor and corrector, such as pairing explicit and backward Euler schemes. Similar attempts have been validated in previous studies (Li et al., 2024; Zhao et al., 2024), where a high-order predictor combined with a single-step backward Euler method demonstrated promising results, particularly on smaller datasets. However, ensuring solution precision inherently requires iterative solvers to obtain the final solution, a process that shares the same merits as high-order methods. Building on this insight, we take a step further to explore whether an iterative corrector mechanism is equally critical for achieving both superior solution fidelity and unlocking genuine efficiency gains.

To this end, we introduce the Iterative Implicit Euler Transformer (IIET). Concretely, in IIET, each iteration represents a computational step within an implicit Euler iterative solver, where multiple corrections to the initial prediction are made to ensure output precision. To further strengthen numerical stability, we also employ linear multi-step methods during each correction step [1]. This architecture, detailed in Figure 1d, is designed not only to achieve superior performance that scales with increasing iterations, exhibiting competitive results against PCformer, but also to be inherently compressible due to its iterative nature. Notably, our top-performing IIET models (340M and 740M parameters) achieve remarkable performance improvements of 2.4% and 2.9% respectively over equivalent vanilla Transformers.

In this way, we can effectively accelerate the inference of IIET via distillation techniques. Here, we further propose Iteration Influence-Aware Distillation (IIAD), a method inspired by structured pruning techniques (Men et al., 2024; Xia et al., 2023; Chen et al., 2024), to reduce dispensable iterations. Specifically, IIAD first assesses "iteration influence" by calculating input-output similarity for each iteration. The optimal number of iterations per layer is then determined according to a predefined influence threshold. Subsequently, a continued pre-training phase is employed to restore the model's capabilities. This process enables users to tailor the iterative correction steps of the IIET model according to their computational budget, yielding efficient IIET variants. Experiments demonstrate that our efficient variant, E-IIET, reduces IIET's inference computational overhead by over 60% while impressively maintaining 99.4% of its performance. The lower bound of 340M and 740M efficient IIET variants not only outperform the vanilla Transformer by 1.9% and 1.3% respectively, but also achieve comparable inference efficiency, showcasing a significant advancement in both performance and deployment efficiency.

## 2 Background

We begin by establishing the connection between residual connections and the Euler method, and then discuss Transformer optimization strategies informed by advanced explicit and implicit numerical solutions of ODEs. Our work builds upon the standard Transformer architecture (Vaswani, 2017), which comprises a stack of identical layers. For language modeling, each layer typically comprises a causal attention (CA) block and a feed-forward network (FFN) block. With residual connections, the output of each block can be formulated as $y_{n+1} = y_n + \mathcal{F}(y_n, \theta_n)$, where $\mathcal{F}(y_n, \theta_n)$ represents the transformation performed by either the CA or FFN block with parameters $\theta_n$.

[1]IIET can be viewed as an instance of the PC paradigm, employing an Euler predictor and an iterative Euler corrector.

### 2.1 Euler Method in Residual Networks

The Euler method provides a linear approximation for first-order ODEs, defined as $y'(t) = f(y(t), t)$ with an initial value $y(t_0) = y_0$. Given a step size $h$ where $t_{n+1} = t_n + h$, the method computes the subsequent value $y_{n+1}$ as:

$$y_{n+1} = y_n + hf(y_n, t_n) \tag{1}$$

where $f(y_n, t_n)$ represents the rate of change of $y$, determined by its current value and time $t$. Notably, this formulation shares a structural similarity with residual networks, where a trainable function, $\mathcal{F}(\cdot)$, approximates these changes. Consequently, from an ODE perspective, residual connections can be interpreted as a first-order discretization of the Euler method. Although the success of residual connections highlights the benefits of the Euler method, its first-order nature introduces significant truncation errors (Li et al., 2022, 2024), limiting the precision of $y_{n+1}$. Fortunately, more advanced numerical methods exist and have been successfully applied to neural networks.

### 2.2 Advanced Numerical Transformers

To improve the precision of $y_{n+1}$, the Runge-Kutta (RK) method offers a more accurate alternative. Inspired by the $o$-order RK method, the ODE Transformer (Li et al., 2022) replaces residual connections with a RK process:

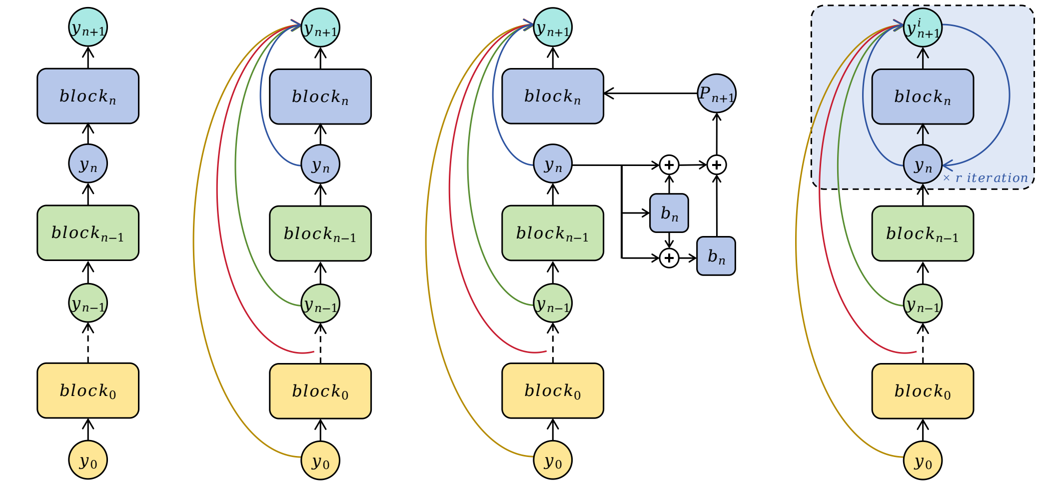
$$y_{n+1} = y_n + \sum\nolimits_{i=1}^{o} \gamma_i \mathcal{F}_i \tag{2}$$

$$\mathcal{F}_1 = \mathcal{F}(y_n, \theta_n) \tag{3}$$

$$\mathcal{F}_i = \mathcal{F}(y_n + \sum\nolimits_{j=1}^{i-1} \beta_{ij}\mathcal{F}_j, \theta_n) \tag{4}$$

where $\mathcal{F}_i$ represents the $i^{th}$ order results computed by a shared transformer block $\mathcal{F}(*, \theta_n)$. The coefficients $\gamma_i, \beta_{ij}$ are learnable parameters. This architecture effectively mitigates truncation error, leading to significant performance gains in generation tasks such as machine translation.

Compared to explicit numerical methods, implicit numerical methods typically offer higher precision and stability. The Predictor-Corrector (PC) method, using an explicit predictor for initial estimates and an implicit corrector for refinement, is a classic example. Recent work has demonstrated the benefits of integrating PC components into neural




(a) Vanilla Transformer (b) Linear Multi-step Transformer (c) Predictor-Corrector Transformer (d) Iterative Implicit Euler Transformer

Figure 1: Architectural comparison: (a) Vanilla Transformer; (b) Linear multistep-enhanced Transformer; (c) PCformer with 2nd-order Runge-Kutta predictor and 1st-order Euler corrector; (d) Our proposed Iterative Implicit Euler Transformer (IIET). The iteration steps $r$ in IIET is configurable, with experimental validation determining $r = 3$ as the optimal setting in this work. All blocks follow an identical computational procedure as the $block_n$.

network architecture. PCformer (Li et al., 2024) employs an $o$-order RK predictor and a linear multistep (Wang et al., 2019) corrector, defined as:

$$y_p = y_n + \sum_{i=1}^{o} \gamma(1-\gamma)^{o-i}\mathcal{F}_i \tag{5}$$

$$y_{n+1} = y_n + \alpha\mathcal{F}(y_p, \theta_n) + \sum_{i=n-2}^{n} \beta\tilde{\mathcal{F}}_i \tag{6}$$

where $\mathcal{F}_i$ shares the same meaning as in Eq. 2 and $\tilde{\mathcal{F}}_i$ denotes the outputs of previous blocks. $\alpha, \beta$, and $\gamma$ are learnable coefficients. Specifically, PCformer's predictor incorporates an Exponential Moving Average (EMA) to weight the contributions of different orders, while the corrector integrates previous block outputs for increased precision. PCformer achieves superior performance over the ODE Transformer and, to some extent, unifies structural paradigms for Transformers improved with implicit numerical methods. Our IIET can be interpreted as a specific instance within the PC paradigm, with a particular emphasis on the iterative corrector component.

## 3 Iterative Implicit Euler Transformer

In this section, we detail the theoretical foundation and core architectural design of the Iterative Implicit Euler Transformer (IIET). Our approach leverages the inherent stability of the implicit Euler method, a cornerstone of numerical analysis, to address key challenges in deep sequence modeling.

### 3.1 Iterative Implicit Euler Method

The implicit Euler method, also known as the Backward Euler method, is a foundational first-order implicit numerical technique celebrated for its robust stability properties, particularly advantageous in handling stiff systems (LeVeque, 2007). Unlike its explicit counterparts, the implicit Euler method employs a backward difference quotient, formulated as:

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}). \tag{7}$$

The implicit nature of Eq. 7, where the computation of $y_{n+1}$ depends on its value at the same time step $t_{n+1}$, inherently requires iterative solvers from numerical analysis to obtain a solution. Specifically, in traditional numerical methods for solving such implicit equations, Newton's iteration is frequently employed due to its quadratic convergence rate and robustness (Zhang et al., 2017; Shen et al., 2020; Kim et al., 2024). However, within the context of neural sequence modeling, where computational efficiency and architectural simplicity are often prioritized, we propose to investigate the efficacy of a simpler alternative: fixed-point iteration (Rhoades, 1976). While prior works like Li et al. (2024) have utilized explicit methods for initial approximations followed by a single Backward Euler correction, the potential of iterative refinement within the implicit corrector remains largely unexplored.

Thus, challenging the implicit assumption that a strong predictor is sufficient for high precision (Li et al., 2024), we propose the central hypothesis that iterative refinement inside the implicit corrector constitutes a pivotal mechanism for enhancing solution fidelity. We argue that a single-step correction inherently limits the achievable accuracy, particularly when modeling intricate sequence dynamics and seeking high-fidelity representations of $y_{n+1}$. Consequently, this work rigorously investigates whether leveraging iterative solutions within the implicit corrector can translate to demonstrable gains in downstream model performance.

Intriguingly, our empirical findings reveal that computationally efficient fixed-point iteration yields surprisingly high precision, often on par with the more computationally intensive Newton's method, particularly within our neural sequence modeling framework. Our proposed Iterative Implicit Euler (IIE) method commences with an initial approximation, $y^0_{n+1}$, derived from an explicit Euler step. This initial estimate is then iteratively refined through $r$ fixed-point iterations as defined below:

$$y^0_{n+1} = y_n + hf(y_n, t_n) \tag{8}$$

$$y^i_{n+1} = y_n + hf(y^{i-1}_{n+1}, t_{n+1}), \quad i \in [1..r]. \tag{9}$$

The final approximation $y_{n+1}$ is thus given by $y^r_{n+1}$, representing the output of the $r^{th}$ iteration.

The IIE method, while formally retaining its first-order numerical accuracy, achieves a significant enhancement in the approximation of $y_{n+1}$ through iterative refinement. This iterative process engenders a structured form of nested computations that superficially resemble higher-order methods, albeit through a fundamentally distinct mechanism rooted in repeated fixed-point iterations. Acknowledging the increased computational cost, the inherent structural regularity of IIE, predicated solely on the preceding iteration's output, emerges as a crucial enabler for inference efficiency optimizations, as detailed in Section 4. This carefully engineered balance between iteratively enhanced precision and structural simplicity underpins the design philosophy of the IIET architecture.

### 3.2 Model Architecture

Building on the IIE method, we propose the Iterative Implicit Euler Transformer (IIET) as a foundational architecture for sequence modeling, particularly for large language models. Adopting the LLaMA architecture (Touvron et al., 2023b) (Transformer++), IIET consists of $N$ stacked transformer decoder layers. Each layer comprises a causal attention module followed by a feedforward module, and employs rotary positional encoding (Su et al., 2024), SiLU activation (Shazeer, 2020), and RMS normalization (Zhang and Sennrich, 2019). Given an input sequence $x = x_1, ..., x_L$ of length $L$, the initial input embeddings are represented as $X^0 = [x_1, ..., x_L] \in \mathbb{R}^{L \times d_{\text{model}}}$, where $d_{\text{model}}$ is the hidden dimension. The output of each subsequent layer is then computed as $X^n = \text{Decoder}(X^{n-1})$, for $n \in [1, N]$.

The key distinction between IIET and Transformer++ lies in IIET's integration of the IIE method within each decoder layer (Figure 1). Unlike Transformer++, which directly computes the layer's output using a single Euler step (standard residual), IIET employs an iterative refinement process. Specifically, IIET first estimates an initial value, $y^0_{n+1}$, via a single Euler step (Eq. 8):

$$y^0_{n+1} = y_n + \mathcal{F}(y_n, \theta_n). \tag{10}$$

where $\mathcal{F}(*, \theta_n)$ represents the $n^{th}$ transformer layer with parameters $\theta_n$. This initial estimate in IIET corresponds to the direct output of each layer in Transformer++.

In the subsequent iterations, our preliminary experiments suggest that incorporating outputs from previous layers, similar to Transformer-DLCL (Wang et al., 2019), can enhance the performance. We thus modify Eq. 9 as follows:

$$y^i_{n+1} = y_n + \alpha_n \mathcal{F}(y^{i-1}_{n+1}, \theta_n) + \sum_{j=0}^{n-1} \alpha_j \tilde{\mathcal{F}}_j, \tag{11}$$

where $i \in [1..r]$ denotes the iteration step, $\tilde{\mathcal{F}}_j$ represents the output of the previous layers $j$, and $\alpha$ represents learnable layer merge coefficients. Appendix A details the computation flow within a single IIET layer.

### 3.3 Experimental Setups

Limited by resources, our experiments primarily explore small-scale language modeling, specifically at parameter scales of 340 million and 740 million.

**Baselines.** We evaluate IIET's performance against two strong baselines: Transformer++ (Touvron et al., 2023a) and PCformer (Li et al., 2024). Transformer++ adopts the LLaMA architecture. PCformer employs a 2nd-order Runge-Kutta predictor and a linear multi-step corrector [2]. All mod-

[2]We also explored a 4th-order Runge-Kutta predictor and more complex correctors, but these increased training costs without substantially improving performance.

| Scale | Model | Wiki. ppl ↓ | LMB. ppl ↓ | LMB. acc ↑ | PiQA acc_norm ↑ | Hella. acc_norm ↑ | SCIQ acc ↑ | ARC-c acc_norm ↑ | Wino. acc ↑ | Avg. ↑ |
|---|---|---|---|---|---|---|---|---|---|---|
| *Pre-training Phase* | | | | | | | | | | |
| *340M Params* | Transformer++ | 28.2 | 78.3 | 28.9 | 64.3 | 34.2 | 76.0 | 23.6 | 51.9 | 46.5 |
| *16B Tokens* | PCformer | 25.7 | 47.0 | 33.1 | 64.9 | 36.3 | 77.5 | **24.7** | **53.3** | 48.3 |
| | IIET | **25.0** | **30.5** | **37.1** | **65.2** | **36.9** | **79.4** | 23.9 | 51.0 | **48.9** |
| *740M Params* | Transformer++ | 23.3 | 34.8 | 36.1 | 66.4 | 38.4 | 78.6 | **24.5** | 50.2 | 49.0 |
| *30B Tokens* | PCformer | 21.2 | 22.0 | 41.0 | 66.3 | 41.3 | 82.0 | 23.3 | 51.2 | 50.9 |
| | IIET | **20.7** | **21.1** | **41.2** | **68.9** | **42.5** | **82.1** | 23.8 | **53.1** | **51.9** |
| *Iteration Influence-Aware Distillation Phase* | | | | | | | | | | |
| *340M Params* | Distil PCformer | 27.2 | 50.4 | 32.2 | **64.6** | 34.9 | 78.0 | 24.7 | 51.3 | 47.6 |
| *5B Tokens* | Lower Bound | 27.0 | 34.6 | 36.1 | 64.0 | 35.0 | **80.7** | 23.0 | 51.5 | 48.4 |
| | E-IIET | **25.7** | **30.9** | **37.4** | 64.4 | **35.8** | 80.4 | **23.5** | **52.1** | **48.9** |
| *740M Params* | Distil PCformer | 22.5 | 29.5 | 37.4 | 66.8 | 39.2 | 80.0 | 23.2 | 50.9 | 49.6 |
| *10B Tokens* | Lower Bound | 23.0 | 29.9 | 37.6 | 67.4 | 38.7 | 79.7 | **25.2** | **53.0** | 50.3 |
| | E-IIET | **21.2** | **24.2** | **40.1** | **68.5** | **41.0** | **81.0** | 24.6 | 52.4 | **51.3** |

Table 1: Comparison of results between our models and baselines in the *Pre-training Phase* and *Iteration Influence-Aware Distillation Phase*. The individual task performance is via zero-shot. We report the main results on the same set of tasks reported by Gu and Dao (2023). The last column shows the average over all benchmarks that use (normalized) accuracy as the metric. **Bold** values represent the best results in each set.

els are trained on the same dataset for an identical token count. Detailed training hyperparameter settings can be found in Appendix B.1.

**Datasets and Evaluation Metrics.** Our models are pre-trained on SlimPajama (Soboleva et al., 2023) and tokenized using the LLaMA2 tokenizer (Touvron et al., 2023a). From the original 627B-token dataset, we sample 16B and 30B tokens for training the 340M and 740M parameter models, respectively. For comprehensive evaluation, we assess perplexity (PPL) on Wikitext (Wiki.) (Merity et al., 2016) and consider several downstream tasks covering common-sense reasoning and question answering: LAMBADA (LMB.) (Paperno et al., 2016), PiQA (Bisk et al., 2020), HellaSwag (Hella.) (Zellers et al., 2019), WinoGrande (Wino.) (Sakaguchi et al., 2021), ARC-Challenge (ARC-c) (Clark et al., 2018), and SCIQ (Welbl et al., 2017). We report PPL on Wikitext and LAMBADA; length-normalized accuracy on HellaSwag, ARC-Challenge, and PiQA; and standard accuracy on the remaining tasks. All evaluations are conducted using the lm-evaluation-harness (Gao et al., 2021).

### 3.4 Experimental Results

**Iteration Steps.** To identify the optimal iteration steps $r$, we first apply varying $r$ values to the 340M IIET model and a smaller 55M parameter variant (detailed in Appendix B.1). All models were evaluated on Wikitext test set. As illustrated in Figure 2, which showcases the benefit of iterative correction,

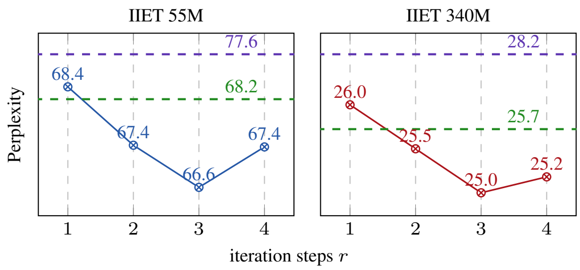



Figure 2: PPL on the Wikitext test set for 55M and 340M IIET across varying iteration steps $r$. Dashed lines indicate Transformer++ and PCformer performance at corresponding parameter scales. Note that IIET's FLOPs is nearly $r + 1$ times of Transformer++.

IIET's performance exceeds PCformer at $r = 2$ and achieves its peak at $r = 3$. Therefore, we adopt $r = 3$ in this work.

**Results.** The advantages of IIET are highlighted by its performance on LLM evaluation benchmarks. As demonstrated in Table 1 *Pre-training Phase*, IIET consistently surpasses Transformer++ and PCformer with comparable capacity. At a parameter scale of 340 million, IIET achieves a mean accuracy of 2.4% higher than that of Transformer++ and 0.6% higher than that of PCformer across all six challenging subtasks. Notably, the performance disparity amplifies progressively with increasing parameter scale, attaining 2.9% and 1% at 740 million parameters. This observation, consistent with Li et al. (2024)'s, confirms the robust scalability of IIET and similar numerical Transformers, showcasing their performance potential with increasing model parameters and training data.

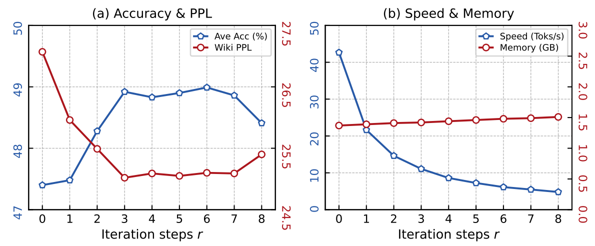



Figure 3: Ablation study on iteration steps $r$: (a) Impact on model performance. (b) Corresponding effects on inference speed and VRAM utilization.

| Model | LMB. | PiQA | Hella. | SCIQ | ARC-c | Wino. | Avg. |
|---|---|---|---|---|---|---|---|
| IIET | 37.1 | 65.2 | 36.9 | 79.4 | 23.9 | 51.0 | 48.9 |
| Trans WS | 30.7 | 63.1 | 34.4 | 75.7 | 23.2 | 50.4 | 46.3 |
| Trans 1.3B | 37.3 | 65.7 | 37.6 | 78.6 | 23.7 | 51.5 | 49.0 |

Table 2: Performance comparison of models with FLOPs comparable to the 340M IIET.

### 3.5 Analysis

**Ablation Study on Iteration Steps.** A key question concerning IIET is whether its performance improves monotonically with an increasing number of iterative correction steps. To investigate this, we conducted an ablation study on the 340M IIET model, varying the number of iteration $r$. [3] As illustrated in Figure 3a, performance initially improves with increasing $r$. However, beyond a certain threshold, further increases in $r$ lead to a plateau in performance gains. This suggests that the iterative refinement process guides the final representation towards a more precise ODE solution, but with diminishing returns after optimal convergence. Detailed downstream results can be found in Appendix C. FurtherMore, to assess the impact of $r$ on inference efficiency, we measured the autoregressive generation throughput of IIET variants on a single A100 GPU. Figure 3b shows that while IIET's inference speed substantially declines with increasing $r$, its VRAM footprint remains largely unaffected as it incurs no extra parameters.

**Comparison with Equal FLOPs.** Given that IIET's iterative correction adds FLOPs (to approximately four times that of Transformer++ when $r = 3$), we aimed for a performance comparison under equivalent computational budgets. Thus, we trained a 1.3B Transformer++ model on identical training data. The results in Table 2 show that IIET performs comparably to the much larger Transformer++ but with substantially fewer parameters, thereby reducing memory and training overhead. Moreover, models with exactly matched parameter scale and FLOPs were benchmarked. Since IIET's architecture closely resembles weight-sharing methods, we established a naive weight-sharing baseline: the Transformer++ model's depth was quadrupled, with weights shared every four layers, namely Trans WS. As shown in Table 2, this simple weight-sharing approach alone does not yield performance gains, highlighting the crucial contribution of IIET's implicit iterative solver-based design to its enhanced performance.

**Parameter Redundancy of IIET.** We hypothesize that the iterative correction process of IIET enhances learning efficiency and reduces parameter redundancy. To investigate this, we used Block Influence (BI) (Men et al., 2024) to measure layer redundancy in IIET and Transformer++. BI assesses the influence of each model block on the hidden state by measuring the similarity between its input and output; lower similarity indicates a higher influence. Specifically, the BI of a Transformer block is calculated as:

$$\mathrm{BI}_i = 1 - \mathbb{E}_{\mathbf{H},t} \frac{\mathbf{H}_{i,t}^T \mathbf{H}_{i+1,t}}{||\mathbf{H}_{i,t}||_2 ||\mathbf{H}_{i+1,t}||_2} \qquad (12)$$

where $\mathbf{H}_{i,t}$ represents the $t^{th}$ row of the $i^{th}$ layer's input hidden states. We randomly sampled 5,000 text segments from Wikitext to calculate the BI of each model. As shown in Figure 4, the influence of IIET's blocks increases significantly with iteration steps, demonstrating higher layer utilization. This also indicates that the learning potential of existing large-scale language models remains under-exploited.

## 4 Iteration Influence-Aware Distillation

While IIET achieves strong downstream task performance, its iterative structure introduces computational overhead that curtails inference speed. This added latency is particularly non-negligible for autoregressive generation in large language models. To enhance IIET's inference efficiency without performance loss, we explore whether continuous pre-training combined with distillation can enable fewer forward passes, ideally a single one, to yield outputs equivalent to those from the complete, multi-step iterative correction process. To this end, we analyze the impact of each iterative correction step on the hidden state within each block. Surprisingly, Figure 5 shows that not all layers require the same number of iteration steps to achieve

[3] In the case where $r = 0$, IIET is structurally the same as the DLCL Transformer.

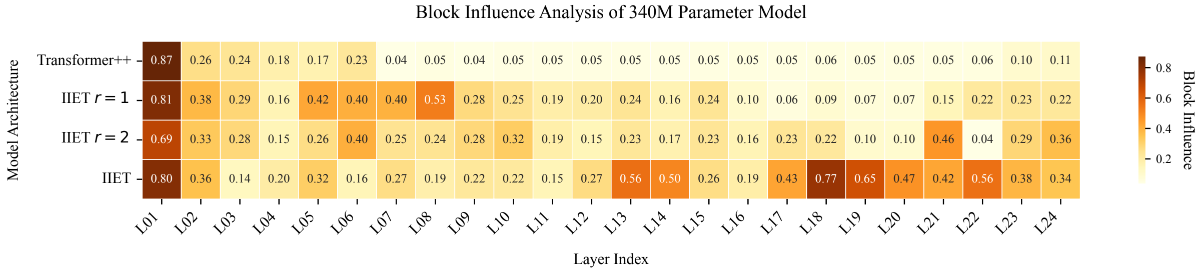



Figure 4: Distribution of Block Influence (BI) for Transformer++ and IIET models with varying iteration steps $r$. Higher BI values indicate lower model redundancy.

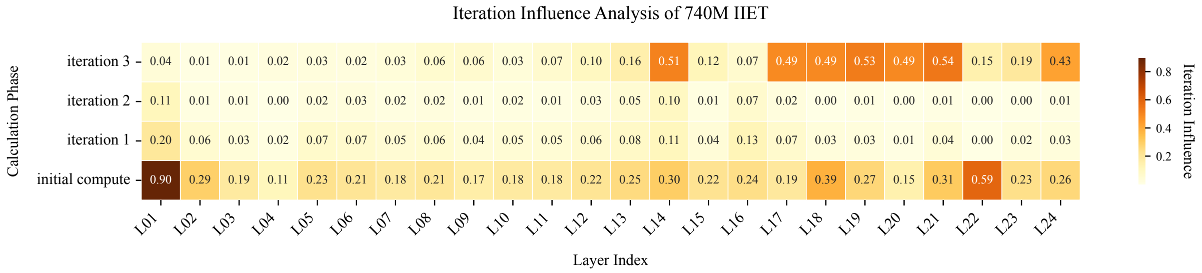



Figure 5: **Iteration Influence** within each layer of the 340M IIET model. Deeper colors indicate larger hidden state changes after this iteration. The 740M IIET results are presented in Appendix D due to space constraints.

accurate output, with deeper layers benefiting more from additional iterative corrections, which is potentially due to the varying roles layers play in the Transformer's representation-building process.

### 4.1 Methodology

In this section, we propose Iteration Influence-Aware Distillation (IIAD). IIAD first analyzes the iterative process of a pre-trained IIET, identifying and eliminating non-essential iterative computations to yield an efficient variant, E-IIET. Subsequently, a layer-wise self-distillation phase restores the performance of E-IIET.

**Iteration Influence.** Iteration influence employs a computational methodology similar to block influence; however, its calculation is performed specifically within individual IIET blocks. For a given $n^{th}$ block, we consider its input $y_n$ and the output $y_{n+1}^i$ of each internal iteration $i$. The pairwise differences between these representations are calculated using Eq. 12 to obtain the iteration influence values. Based on these values and a specified computational budget, users can determine the number of iteration steps to retain per block.

In this work, we primarily investigate two designs for efficient IIET variants: ❶ **Lower Bound**: Each layer performs only a single forward pass, establishing a performance lower bound for efficient IIET. ❷ **E-IIET**: This variant establishes a threshold using the minimum of the initial iteration influence values computed in each layer. Consequently, iteration steps with influence scores below this threshold are omitted, preserving each layer's initial computation and essential iteration steps. Specifically, E-IIET reduces the number of iteration steps from a baseline of 72 to 15 in the 340M variant and 23 in the 740M variant.

**Iteration Influence-Aware Distillation.** In the continuous pre-training stage, we employ a warm-start initialization strategy, directly inheriting parameters from the pre-trained IIET model to retain knowledge acquired during its initial pre-training phase. To enable efficient IIET variants (e.g., E-IIET) to approximate the precise output representations of the full IIET, we utilize a fine-grained, block-specific knowledge distillation framework incorporating two complementary losses: **1) Mean Squared Error (MSE) Loss**: For each block, an MSE loss encourages E-IIET to mimic the refined hidden states produced by the full IIET. This loss is computed as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{h}_i^{\text{IIET}} - \mathbf{h}_i^{\text{E-IIET}}\|_2^2 \qquad (13)$$

where $\mathbf{h}_i$ are the hidden state outputs of the $i^{th}$ block. **2) Kullback-Leibler (KL) Loss**: To further align prediction behavior, we compute the KL divergence between the final output probability distributions of the full IIET and E-IIET:

$$\mathcal{L}_{\text{KL}} = D_{\text{KL}} \left( p(\mathbf{z}^{\text{IIET}}/\tau) \parallel p(\mathbf{z}^{\text{E-IIET}}/\tau) \right) \qquad (14)$$

| Model | 340M | | | 740M | | |
|---|---|---|---|---|---|---|
| | Spd. | FLOPs | VARM | Spd. | FLOPs | VARM |
| Transformer++ | 49.97 | 0.38 | 1.37 | 48.91 | 0.80 | 2.80 |
| PCformer | 14.14 | 1.06 | 1.41 | 14.38 | 2.30 | 2.86 |
| IIET | 11.07 | 1.40 | 1.42 | 10.95 | 3.05 | 2.89 |
| Lower Bound | 42.66 | 0.38 | 1.37 | 42.03 | 0.80 | 2.80 |
| E-IIET | 25.95 | 0.60 | 1.38 | 22.12 | 1.52 | 2.83 |

Table 3: A comparison of inference speed (tokens per second), FLOPs (T) and VARM (GB) for baseline models, PCformer, and efficient IIET variants.

where $\mathbf{z}$ represent the output logits and $\tau$ is the distillation temperature. By combining these two distillation losses with Cross-Entropy loss, we train E-IIET to effectively capture the knowledge embedded within the full IIET's iterative refinement process. The final training objective for this continuous pre-training stage is thus:

$$\mathcal{L}_{\text{E-IIET}} = \mathcal{L}_{\text{CE}} + \alpha\mathcal{L}_{\text{MSE}} + \beta\mathcal{L}_{\text{KL}} \tag{15}$$

### 4.2 Experiments and Results

**Setups.** To train efficient IIET variants, we sample one-third of the original pre-training tokens (see Appendix B.2 for detailed training settings). For performance comparison against E-IIET, we also prepare two key baselines: a *Lower Bound* variant, which omits all iterative corrections, and a distilled version of PCformer. All models are trained following the method outlined in Section 4.1.

**Main Results.** Table 1 presents the main results for IIAD. As a baseline, directly distilling PCformer into a standard Euler architecture (namely *Distil PCformer*) leads to substantial performance degradation, highlighting the importance of the sophisticated numerical solvers employed by higher-order methods to achieve their accuracy. In contrast, E-IIET, compared to the full IIET model, retains the vast majority of its performance while reducing the average iterative correction overhead by about 55%. Importantly, even the *Lower Bound* efficient IIET variant achieves performance on par with PCformer, demonstrating IIET's strength in balancing efficiency with strong performance.

**Inference Efficiency.** We analyze the inference speed, FLOPs and VRAM usage of our main models. As Table 3 indicates, E-IIET achieves over a 2x speedup compared to full IIET, while largely maintaining IIET's performance advantage (E-IIET vs. full IIET scores: 48.9/48.9 for 340M and 51.3/51.9 for 740M model). However, due to the FLOPs incurred by its remaining iteration steps, E-IIET still exhibits nearly twice the inference latency of Transformer++. A key characteristic of these efficient IIET variants is the inverse relationship between performance and efficiency: fewer iterations lead to lower performance but higher efficiency. Notably, Table 3 shows that the maximum efficiency attained by these variants (i.e, *Lower Bound*) is close to that of the Transformer++, with their average performance surpassing it by 1.6 points. This adaptability makes E-IIET a flexible solution for practical deployment, as users can select the iteration steps based on their resource constraints (e.g., reducing iterations to maximize inference speed).

## 5 Related Work

The link between residual connections and ODEs, first established by Weinan (2017), has spurred extensive research into ODE-based neural network architectures. This insight has paved the way for applying ODE techniques to benefit diffusion models (Liu et al., 2022) and Transformers (Li et al., 2022). For instance, DPM-Solver (Lu et al., 2022a,b) accelerates diffusion model sampling by employing exact ODE solution formulations and higher-order numerical techniques; Kim et al. (2024) and Li et al. (2020) focus on implicit Euler methods for improved adversarial robustness. In this work, we distinctively focus on using implicit Euler methods to enhance language model performance. Other works have designed novel ODE-based architectures (Chen et al., 2018). DEQ (Bai et al., 2019), for example, replaces sequences of explicit layers with a single implicit layer solved via equilibrium finding. In contrast, our approach enhances explicitly defined layers by employing sophisticated implicit numerical solvers for their forward pass computation. More recently, PCformer (Li et al., 2024) has demonstrated significant gains in language modeling and machine translation. However, our proposed IIET exceeds PCformer in performance, features a simpler architecture, and achieves superior inference efficiency.

## 6 Conclusions

We introduce the Iterative Implicit Euler Transformer (IIET), which leverages an iterative implicit Euler method to achieve superior and scalable performance over both vanilla Transformers and PCformer. Furthermore, we develop an inference acceleration technique for IIET that allows users to adjust inference efficiency based on their budget.

## 7 Limitations

Computational resource constraints currently preclude a comprehensive evaluation of IIET on larger-scale language models. Furthermore, while our IIAD method is designed to produce efficient IIET variants for inference, the IIAD process itself introduces notable computational overhead during its application. Future research will focus on integrating the determination of layer-specific iteration requirements directly into the pre-training stage. This could facilitate the direct training of inherently efficient IIET models, potentially bypassing a separate, resource-intensive distillation phase.

Beyond optimizing IIET's per-token efficiency, we also identify a promising avenue for broader application. Current large reasoning models often achieve high performance by generating substantially more tokens than are present in the final answer, leading to significant inference latency. IIET, on the contrary, enhances per token representational power through depth-wise iterative refinement, albeit at an increased per-token computational cost. We hypothesize that this trade-off could be ultimately advantageous in multi-step reasoning tasks: IIET's more precise computation per token might enable it to generate complete and correct answers in fewer overall autoregressive steps, thereby reducing the total token count and potentially overall latency. Validating this hypothesis, however, necessitates training and evaluating IIET at larger model and data scales, which remains a key direction for future investigation.

## References


Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. *Advances in neural information processing systems*, 32.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, and 1 others. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. 2018. Neural ordinary differential equations. *Advances in neural information processing systems*, 31.

Xiaodong Chen, Yuxuan Hu, and Jing Zhang. 2024. Compressing large language models by streamlining the unimportant layer. *arXiv preprint arXiv:2403.19135*.

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

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, and 1 others. 2021. A framework for few-shot language model evaluation. *Version v0. 0.1. Sept*, 10:8–9.

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

Geoffrey Hinton. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851.

Mihyeon Kim, Juhyoung Park, and Youngbin Kim. 2024. Im-bert: Enhancing robustness of bert through the implicit euler method. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16217–16229.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. *arXiv preprint arXiv:1606.07947*.

Randall J. LeVeque. 2007. *Finite difference methods for ordinary and partial differential equations - steady-state and time-dependent problems*. SIAM.

Bei Li, Quan Du, Tao Zhou, Yi Jing, Shuhan Zhou, Xin Zeng, Tong Xiao, JingBo Zhu, Xuebo Liu, and Min Zhang. 2022. Ode transformer: An ordinary differential equation-inspired model for sequence generation. *arXiv preprint arXiv:2203.09176*.

Bei Li, Tong Zheng, Rui Wang, Jiahao Liu, Qingyan Guo, Junliang Guo, Xu Tan, Tong Xiao, Jingbo Zhu, Jingang Wang, and 1 others. 2024. Predictor-corrector enhanced transformers with exponential moving average coefficient learning. *arXiv preprint arXiv:2411.03042*.

Mingjie Li, Lingshen He, and Zhouchen Lin. 2020. Implicit euler skip connections: Enhancing adversarial robustness via numerical stability. In *International Conference on Machine Learning*, pages 5874–5883. PMLR.

Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. 2022. Pseudo numerical methods for diffusion models on manifolds. *arXiv preprint arXiv:2202.09778*.

Ilya Loshchilov, Frank Hutter, and 1 others. 2017. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2022a. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2022b. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*.

Xin Men, Mingyu Xu, Qingyu Zhang, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

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*.

BE Rhoades. 1976. Comments on two fixed point iteration methods. *Journal of Mathematical Analysis and Applications*, 56(3):741–750.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Noam Shazeer. 2020. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*.

Jiawei Shen, Zhuoyan Li, Lei Yu, Gui-Song Xia, and Wen Yang. 2020. Implicit euler ode networks for single-image dehazing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 218–219.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. Slimpajama: A 627b token cleaned and deduplicated version of redpajama.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Anh Tong, Thanh Nguyen-Tang, Dongeun Lee, Duc Nguyen, Toan Tran, David Leo Wright Hall, Cheongwoong Kang, and Jassik Choi. 2025. Neural ode transformers: Analyzing internal dynamics and adaptive fine-tuning. In *ICT.R.2025 Poster*. Unpublished.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023a. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. 2019. Learning deep transformer models for machine translation. *arXiv preprint arXiv:1906.01787*.

Ee Weinan. 2017. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 1(5):1–11.

Johannes Welbl, Nelson F Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2023. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

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

Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. 2017. Polynet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 718–726.

Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. 2024. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36.

Kaiwen Zheng, Cheng Lu, Jianfei Chen, and Jun Zhu. 2024. Dpm-solver-v3: Improved diffusion ode solver with empirical model statistics. *Advances in Neural Information Processing Systems*, 36.

# A IIET Algorithm

Algorithm 1 details the computation flow within a single IIET layer, where $\mathbf{L}$ stores the hidden states from previously computed layers, providing necessary context. During the computation within a single block, an initial estimate of its output, $y^0_{n+1}$, is iteratively refined. Each iteration i updates this estimate to $y^i_{n+1}$ through the function $\mathcal{F}$ and the context $\mathbf{L}$. This fixed-point iteration process progressively converges towards a more precise final output $y_{n+1}$.

**Algorithm 1** Iterative Implicit Euler Paradigm

**procedure** IIET BLOCK($\mathbf{y_n}$, $\mathbf{L}$)
  $f^0_n \leftarrow \mathcal{F}(\mathbf{y_n}, \theta_\mathbf{n})$ ▷ Compute initial value
  $\mathbf{L}$.append($f^0_n$) ▷ Store $f^0_n$
  **for** $i \leftarrow 0$ **to** $r-1$ **do**
    Compute $\mathbf{y^i_{n+1}}$ using $\mathbf{L}$ via Eq. 11
    $f^{i+1}_n \leftarrow \mathcal{F}(\mathbf{y^i_{n+1}}, \theta_\mathbf{n})$ ▷ Compute correct value
    $\mathbf{L}$.update($f^i_n \rightarrow f^{i+1}_n$) ▷ Update $f^i_n$
  **end for**
  Compute $\mathbf{y^r_{n+1}}$ using $\mathbf{L}$ via Eq. 11
  **return** $\mathbf{y^r_{n+1}}$ ▷ Return the layer output
**end procedure**

# B Training Settings

## B.1 Pre-training Phase

For our main experiments, all models are trained from scratch at two parameter scales (340M and 740M) to evaluate IIET's performance across different sizes. We utilize the AdamW (Loshchilov et al., 2017) optimizer with a maximum learning rate of 3e-4 for all models. Batch sizes are set to 0.5M tokens for 340M models and 1M tokens for 740M models. A cosine learning rate schedule is applied to both model scales, featuring a 0.01 warmup ratio, 0.01 weight decay, and gradient clipping at 1.0. Furthermore, to identify the optimal iteration count $r$, we train a dedicated, smaller IIET model variant with just 55M parameters. All hyperparameter specifications for the pre-training phase are available in Table 4.

## B.2 Iteration Influence-Aware Distillation Phase

To train efficient IIET variants, we sample one-third of the total pre-training tokens for each configuration (e.g., 5 billion token for 340M models and 10 billion token for 740M models). Users can customize the corrective iteration process for these variants based on their computational budget. In this study, we focus on two main types of efficient IIETs: a 'lower bound' configuration that removes all iterative steps, and E-IIET, which utilizes a threshold for iteration selection.

For training, all efficient IIET variants use the full IIET as a teacher model and are trained with the fine-grained supervision method detailed in Section 4.1. We apply a cosine decay learning rate schedule with an initial value of 2e-4, while other pre-training hyperparameters are kept consistent. Furthermore, for comparison purposes, we train Distil PCformer, a self-distilled version of PCformer using the same methodology. To ensure a fair comparison, we use the same evaluation dataset and metrics described in Section 3.3.

# C IIET with Varying Iteration Steps

We evaluated the downstream task performance of our 340M model across iteration steps $r = 0$ to $r = 8$, as detailed in Section 3.5. Table 5 shows that as the number of iterations increases, IIET's performance on downstream tasks initially improves progressively before these gains begin to plateau. Although performance slightly degrades at $r = 8$, IIET still surpasses both Transformer++ and PCformer. Notably, with $r = 2$ iterations, IIET achieves performance comparable to PCformer with its per-block forward pass count is also similar to PCformer's. This demonstrates that our proposed iterative implicit Euler (IIET) architecture, despite its simpler design, offers representation refinement capabilities that are close to those of higher-order methods. Finally, using identical training data, IIET exhibited superior data-fitting ability over the other models, as indicated by its perplexity (PPL) scores.

# D Iteration Influence of 740M IIET

Figure 6 displays the Iteration Influence of the 740M IIET model. By selecting the minimum initial computation of each layer as the threshold, we can reduce the number of corrective iterations from 72 to 23.

| **Hyperparameters** | **55M** | **340M** | **740M** |
|---|---|---|---|
| model_type | llama | llama | llama |
| hidden_act | silu | silu | silu |
| initializer_range | 0.02 | 0.02 | 0.02 |
| hidden_size | 512 | 1024 | 1536 |
| intermediate_size | 1408 | 2816 | 4224 |
| max_position_embeddings | 2048 | 2048 | 2048 |
| num_attention_heads | 4 | 8 | 8 |
| num_hidden_layers | 12 | 24 | 24 |
| num_key_value_heads | 4 | 8 | 8 |
| pretraining_tp | 1 | 1 | 1 |
| rms_norm_eps | $1.00 \times 10^{-6}$ | $1.00 \times 10^{-6}$ | $1.00 \times 10^{-6}$ |
| tie_word_embeddings | True | True | True |
| torch_dtype | float16 | float16 | float16 |
| vocab_size | 32000 | 32000 | 32000 |
| training_len | 2048 | 2048 | 2048 |
| total_batch_size | 128 | 256 | 512 |
| learning_rate | 0.0004 | 0.0003 | 0.0003 |
| max_steps | 5000 | 30000 | 30000 |
| warm_up | 0.05 | 0.05 | 0.01 |

Table 4: Model Hyperparameters and Training Hyperparameters.

| **Model** | **Wiki.** ppl ↓ | **LMB.** ppl ↓ | **LMB.** acc ↑ | **PiQA** acc_norm ↑ | **Hella.** acc_norm ↑ | **SCIQ** acc ↑ | **ARC-c** acc_norm ↑ | **Wino.** acc ↑ | **Avg.** ↑ |
|---|---|---|---|---|---|---|---|---|---|
| Transformer++ | 28.23 | 78.31 | 28.93 | 64.31 | 34.23 | 76.00 | 23.63 | 51.93 | 46.51 |
| PCformer | 25.71 | 47.02 | 33.10 | 64.92 | 36.31 | 77.53 | 24.70 | 53.26 | 48.30 |
| IIET $r = 0$ | 27.07 | 48.52 | 32.43 | 65.07 | 34.80 | 78.30 | 23.46 | 50.36 | 47.40 |
| IIET $r = 1$ | 25.96 | 36.34 | 34.43 | 64.69 | 36.07 | 76.30 | 23.29 | 50.12 | 47.48 |
| IIET $r = 2$ | 25.49 | 35.76 | 34.64 | 64.96 | 36.80 | 77.20 | 24.23 | 51.85 | 48.28 |
| IIET $r = 3$ | 25.02 | 30.51 | 37.05 | 65.23 | 36.93 | 79.40 | 23.89 | 50.99 | 48.92 |
| IIET $r = 4$ | 25.09 | 29.94 | 36.79 | 64.31 | 37.25 | 78.10 | 22.78 | 53.75 | 48.83 |
| IIET $r = 5$ | 25.05 | 30.58 | 36.32 | 64.51 | 37.34 | 78.55 | 23.42 | 53.23 | 48.90 |
| IIET $r = 6$ | 25.10 | 31.21 | 35.84 | 64.71 | 37.43 | 79.00 | 24.06 | 52.91 | 48.99 |
| IIET $r = 7$ | 25.09 | 31.62 | 35.50 | 65.16 | 36.98 | 79.20 | 23.34 | 52.98 | 48.86 |
| IIET $r = 8$ | 25.40 | 32.02 | 35.16 | 65.61 | 36.52 | 79.40 | 22.61 | 51.14 | 48.41 |

Table 5: Performance comparison of IIET with varying iteration steps at 340 million parameters.

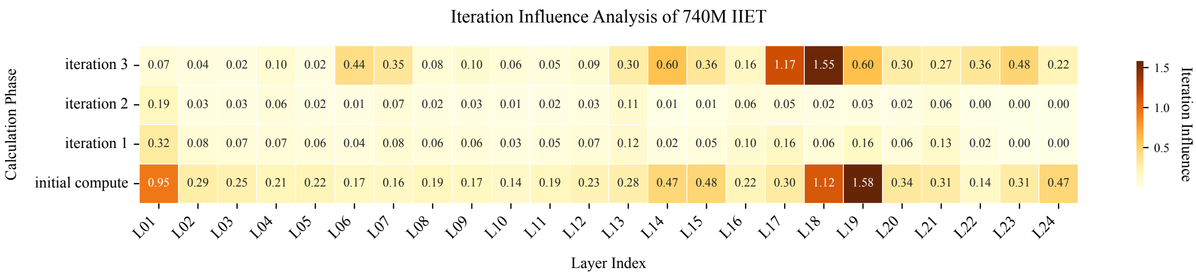



Figure 6: Impact of different iteration stages on the hidden state within each layer of the 740M IIET model, which we term **iteration influence**. Deeper colors indicate larger hidden state changes after this iteration.