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 continued training phase, IIAD eliminates non-essential iterations, reducing IIET's inference computational overhead by over 60% while maintaining 99.4% task performance accuracy. On lm-evaluation-harness, IIET demonstrates a 2.65% improvement over vanilla Transformers and a 0.8% gain over PCformer in average accuracy. The efficient variant E-IIET achieves a 1.83x speedup and a performance gain exceeding 0.5% compared to PCformer.

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 PCformers (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 PCformers impedes effective knowledge transfer via distillation. Our empirical investigations reveal a obvious 54% performance degradation in distilled student models, even the students initialized with PCformer's 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 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 multistep methods during each correction step [1]. This architecture, detailed in Figure 1 (d), is designed to not only achieve superior performance that scales with increasing iterations, exhibiting competitive results against PCformers, 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 disitillation techniques. Here, we further propose an Iteration Influence-Aware Distillation (IIAD), inspired by structured pruning techniques (Men et al., 2024; Xia et al., 2023; Chen et al., 2024), to reduce redundant iterations. Specifically, IIAD first evaluates 'iteration influence' by measuring the similarity between the inputs and outputs of each iteration, determining the optimal number of iterations per layer. Subsequently, a continued pre-training phase is employed to restore the model's capabilities. This process yields efficient IIET (E-IIET), which demonstrably reduces IIET's inference computational overhead by over 60% while impressively maintaining 99.4% of its downstream task performance. Ultimately, our 340M and 740M parameter E-IIET models not only outperform the vanilla Transformer by 2.4 and 2.3 points, respectively but also achieve a 1.83x speedup with a performance gain exceeding 0.5% compared to PCformers, 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 θ_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:

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

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

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

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

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 network architecture. PCformer (Li et al., 2024)

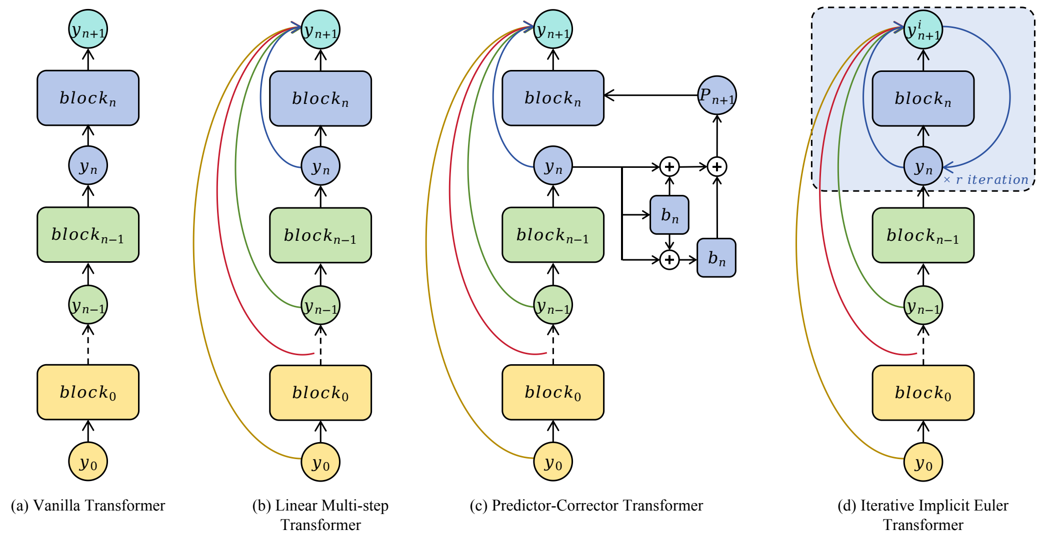

Figure 1: Architectural comparison: (a) Vanilla Transformer; (b) Linear multistep-enhanced Transformer; (c) PCformer with 2nd-order Runge-Kutta predictor and 1st-order Euler corrector; (d) Our proposed Iterative Implicit Euler Transformer (IIET). The iteration steps r in IIET is configurable, with experimental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

employs an o-order RK predictor and a linear multistep (Wang et al., 2019) corrector, defined as:

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

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

where $\mathcal{F}_i$ shares the same meaning as in Eq. 2 and $\tilde{\mathcal{F}}_i$ denotes the outputs of previous blocks. α, β, and γ 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:

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

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. 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 (Rhoades, 1976) 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, superficially resembling 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_{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 next layer's output using a single Euler step, 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 θ_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 α represents learnable layer merge coefficients. Appendix A.1 details the computation flow within a single IIET layer.

3.3 Experimental Setups

Limited by resources, our experiments primarily explore small-scale language modeling, benchmarking IIET against a competitive Transformer baseline incorporating modern architectural improvements, as well as the PCformer, which employs an advanced high-order method as a predictor and a multistep method as a corrector.

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 sampled 16B and 30B tokens for training the 340M and 740M parameter models, respectively. For comprehensive evaluation, we assess perplexity (PPL) on Wikitext

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	B-PCformer	27.2	50.4	32.2	**64.6**	34.9	78.0	24.7	51.3	47.6
5B Tokens	B-IIET	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	B-PCformer	22.5	29.5	37.4	66.8	39.2	80.0	23.2	50.9	49.6
10B Tokens	B-IIET	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 baseline models 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.

(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).

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, incorporating rotary positional embeddings, SiLU activation, and RMS normalization. PCformer employs a 2nd-order Runge-Kutta predictor and a linear multi-step corrector.[2] For a fair comparison, all models were trained on the same dataset for an identical number of tokens.

Training Details. We train all models from scratch at two scales, 340M and 740M parameters, to assess IIET's performance across different sizes. All models are trained using AdamW (Loshchilov et al., 2017) with a maximum learning rate of 3e-4.

[2]We also explored using a 4th-order Runge-Kutta predictor and more complex correctors, but these resulted in increased training costs without substantial performance improvements in language modeling.

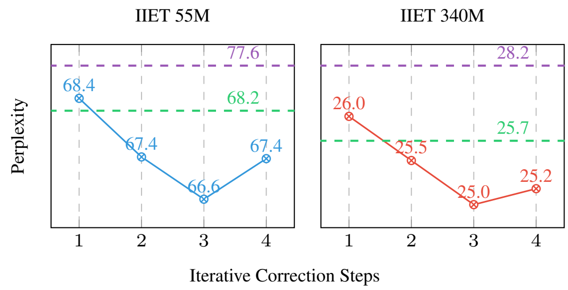

Figure 2: Perplexity (PPL) on the Wikitext test set for the 55M and 340M IIET models as a function of iterative correction steps. The purple and green dashed lines represent the PPL of Transformer++ and PCformer, respectively, at the same parameter scale.

The 340M models use a batch size of 0.5M tokens, while the 740M models use a batch size of 1M tokens. We employ a cosine learning rate schedule with a warmup ratio of 0.01, a weight decay of 0.01, and a gradient clipping of 1.0 for both model sizes.

3.4 Experimental Results

Number of Iterations. We initiated the process by identifying the optimal iteration steps r using the 340M IIET model and a smaller variant with only 55 million parameters (detailed in Appendix A.3), and then applying the optimized value to larger models. We evaluated the models' performance on Wikitext. The performance gains, as illustrated in Figure 2, demonstrate the benefit of iterative correction. Specifically, IIET's performance exceeds PCformer at $r = 2$ and reaches its peak at $r = 3$.

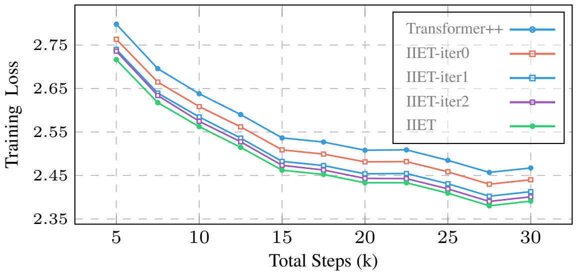

Figure 3: A comparison of training curves between Transformer++ and IIET with different iteration counts at the 340M parameter scale.

Model	340M		740M	
	Inference	Memory	Inference	Memory
Transformer++	49.97	1.37	48.91	2.80
PCformer	14.14	1.41	14.38	2.86
IIET	11.07	1.42	10.95	2.89
IIET-iter0	42.66	1.37	42.03	2.80
IIET-iter1	21.64	1.39	21.47	2.83
IIET-iter2	14.63	1.41	14.33	2.86
E-IIET	25.95	1.38	22.12	2.83

Table 2: A comparison of inference speed (tokens per second), memory consumption (GB) for baseline models, IIET with varying iteration counts, and E-IIET at the 340M and 740M parameter scales.

Results. The advantages of IIET are highlighted by its performance on large language model 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.

3.5 Analysis

Impact of Iteration Steps. To further analyze the impressive performance of IIET, we conducted experiments with the 340 million parameter model. Figure 3 shows the training curves for IIET with different iteration steps. As the number of iterations increases, the model's ability to fit the data gradually improves. The evaluation on the benchmark confirms that IIET, with 1, 2, and 3 iterations, consistently outperforms Transformer++ across all tasks (Appendix A.2), which validates the effectiveness of the proposed method.

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 Influenc (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 higher influence. Specifically, the BI of a Transformer block is calculated as:

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

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.

Inference Efficiency. While IIET achieves strong downstream task performance, the additional computation introduced by the iterative structure also limits its inference speed. For autoregressive generation in large language models, the additional latency during inference is non-negligible. Using a single A100 GPU, we compared the generation speed and memory usage of various large model configurations during autoregressive inference. As shown in Table 2, while maintaining a comparable memory footprint to Transformer++ at the same parameter scale, Transformers enhanced through numerical methods exhibit proportionally higher inference latency. This increased latency arises from the computational complexity of the numerical solvers used for higher accuracy. Similarly, for IIET models, the observed increase in latency is directly proportional to the number of inference iterations.

4 Iteration Influence-Aware Distillation

To improve IIET's inference efficiency without sacrificing performance, we explored the potential of continuous pre-training to enable a single forward

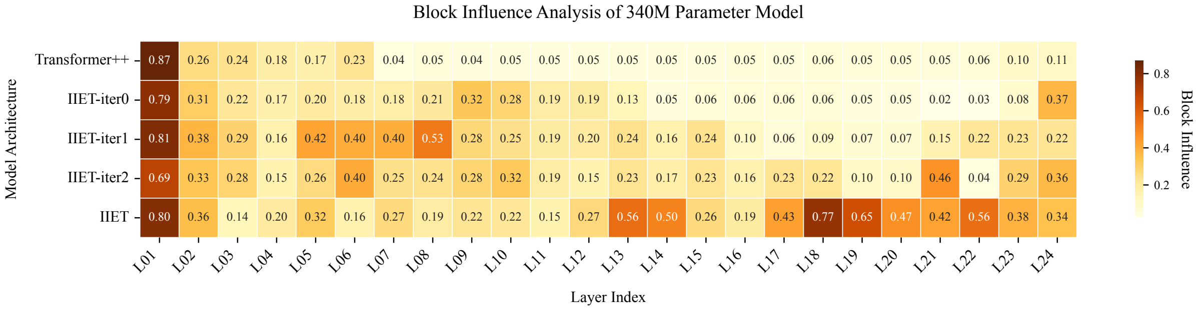

Figure 4: Block Influence (BI) distribution across different model architectures at the 340M parameter scale. Higher BI values indicate lower model redundancy.

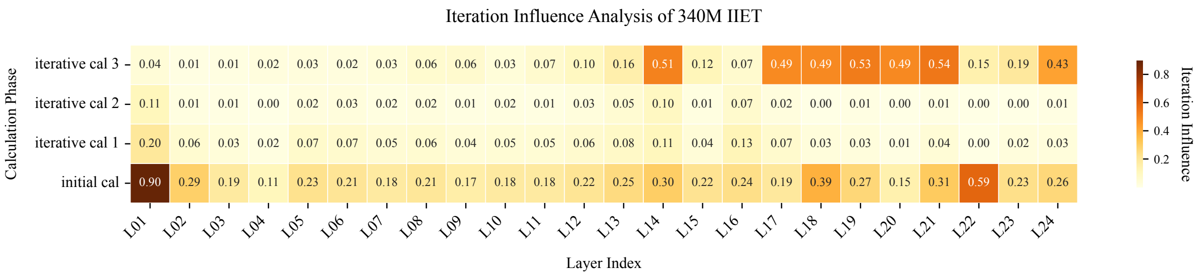

Figure 5: Impact of different iteration stages on the hidden state within each layer of the 340M IIET model, which we term **iteration influence**. Deeper colors indicate larger hidden state changes after this iteration. Due to space constraints, the results for 740M models will be included in the appendix A.4.

pass that produces outputs equivalent to multiple iterative corrections. Although a warm-start knowledge distillation approach was initially considered, our findings (Section 4.3) indicate its difficulty in achieving model reconvergence to an optimal point. Recognizing that increased computational capacity is crucial for maximizing parameter utilization in IIET, we hypothesized that the varying roles of layers in representation building within the Transformer architecture imply that *not all layers require the same iteration steps for accurate output.* To validate this, we analyzed the impact of each iteration on the hidden state within each block. As shown in Figure 5, we observed significant variations in this impact across different layers, with deeper layers appearing to benefit more from additional iterations.

4.1 Methodology

To enhance IIET's inference efficiency, we propose Iteration Influence-Aware Distillation (IIAD). IIAD analyzes the iterative process of a pre-trained IIET, identifying and eliminating redundant computations to produce an efficient IIET (E-IIET). A subsequent layer-wise self-distillation phase then recovers E-IIET's performance.

Iteration Influence. Iteration influence follows the same computational methodology as block influence, differing in that its calculation is conducted within each individual IIET block. For the n^{th} block, we consider the input y_n and the representations y_{n+1}^i from each iteration [3]. Using Eq. 12, we compute the pairwise differences between these representations. We hypothesize that iterations with an Iteration Influence below 0.1 are redundant. As shown in Figure 5, a threshold of 0.1 allows the removal of most iterations while preserving the initial computation within each block. Based on this criterion, we statically determine the minimum number of iterations required per layer, reducing IIET's computational cost without affecting the number of model parameters. Specifically, the total number of iterative correction was reduced from 72 to 15 (340M) and 23 (740M). We refer to this optimized structure as E-IIET.

Iteration Influence-Aware Distillation. E-IIET's continuous pre-training stage employs a warm-start initialization strategy, directly inheriting parameters from the pre-trained IIET model to retain the knowledge acquired during the initial pre-training phase. To enable E-IIET to approximate the precise output representations of IIET, we employ a fine-grained, block-specific knowledge distillation framework via two complementary losses: **1) Mean Squared Error (MSE) Loss**: For each block, we use an MSE loss to make

[3]Note that the final iteration's result is the current IIET block's output.

E-IIET mimic the refined hidden states produced by IIET. This loss is computed as:

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

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

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

where $\mathbf{z}$ represents the logits and τ is the temperature coefficient. By combining these two loss functions, we train E-IIET to effectively capture the knowledge embedded within IIET's iterative refinement process. The final training objective for continued pre-training is:

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

4.2 Experiments

To train E-IIET, we sample one-third of the total pre-training tokens for each configuration. We employ a cosine decay learning rate schedule with a initial value of 2e-4, while maintaining all other pre-training hyperparameters. To evaluate the effectiveness of IIAD, we compare E-IIET against two baseline student models with standard Euler structure: B-IIET (initialized with IIET parameters) and B-PCformer (initialized with PCformer parameters). All the baselines are trained using the fine-grained supervision method detailed in Section 4.1. For a fair comparison, we used the same evaluation dataset and metrics described in Section 3.3.

4.3 Results

Main Results. Table 1 shows the main results of *Iteration Influence-Aware Distillation Phase*. Direct distillation of PCformer and IIET into a standard Euler structure leads to significant performance degradation (i.e. B-PCformer, B-IIET), highlighting the critical role of the additional computational budget in higher-order methods for maintaining computational accuracy. Compared to IIET's performance in the *Pre-training Phase*, E-IIET retains most of the model's capabilities while reducing the average additional iterative computational overhead by 70%. This demonstrates the effectiveness of the IIAD method.

Inference Efficiency. We compared the inference speed and memory usage of our main models on two parameter scales. Table 2 shows that E-IIET achieves over a 2x speedup and improved memory efficiency compared to IIET. However, due to the additional FLOPs introduced by the necessary iterative process, E-IIET still experiences some inference latency compared to the vanilla Transformer. In future work, we expect to further explore the possibility of improving IIET efficiency by leveraging techniques such as conditional computation.

5 Related Work

The connection between residual connections and ODEs, initially proposed by Weinan (2017), has spurred extensive research into ODE-based neural network architectures. This includes innovative designs like Neural ODEs (Chen et al., 2018) and applications to convolutional networks (Zhu et al., 2023). Several works based on implicit Euler methods have focused on enhancing model adversarial robustness and generalization capabilities (Kim et al., 2024; Li et al., 2020), whereas we concentrate on improving language model performance. Recent efforts have successfully applied ODE principles to Transformers, exemplified by PCformer (Li et al., 2024), which shows substantial improvements in language modeling tasks. Our proposed IIET, however, achieves stronger performance with a simpler architecture and offers enhanced inference efficiency compared to PCformer.

6 Conclusions

We propose a novel Transformer architecture, the Iterative Implicit Euler Transformer (IIET), designed for enhanced language modeling performance. IIET leverages the iterative implicit Euler method, providing substantial improvements over vanilla Transformers with a simplified architecture compared to PCformer. Our experiments show that IIET's performance advantage over both baselines grows with model size, with significant gains observed at 340M and 740M parameters. Furthermore, we introduce an inference acceleration technique for IIET, which employs iteration influence analysis and continued pretraining to reduce redundant computations. This approach achieves a 2× inference speedup while preserving the model's performance benefits.

7 Limitations

Limitations in computational resources precluded the evaluation of IIET's performance on larger language models. Additionally, the IIAD method, designed to improve efficiency over IIET, introduces further computational demands. Future research will focus on exploring the feasibility of determining layer-specific iteration requirements during pretraining, thus facilitating the creation of efficient IIET models through single-pass training.

References

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 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, et al. 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, et al. 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.

Ilya Loshchilov, Frank Hutter, et al. 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, et al. 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, et al. 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.

Mai Zhu, Bo Chang, and Chong Fu. 2023. Convolutional neural networks combined with runge–kutta methods. *Neural Computing and Applications*, 35(2):1629–1643.

A Appendix

A.1 IIET Algorithm

Algorithm 1 details the computation flow within a single IIET layer, where $\mathbf{H}$ stores the previously computed.

Algorithm 1 Iterative Implicit Euler Paradigm

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

A.2 Performance details of the model with different iterations.

We evaluated the downstream task performance of the 340M model trained with varying numbers of iterations, as described in Section 3.5. As shown in Table 3, with an increase in the number of iterations, the performance of IIET on downstream tasks progressively improved. It achieved comparable performance to PCformer at two iteration corrections. Using identical training data, IIET showed superior data fitting ability, as indicated by its perplexity (PPL) scores, compared to other models.

A.3 55M IIET model

We train the IIET model with 55 million parameters to validate the optimal iteration steps. The model is stack of 12 layers with a hidden dimension 512. The IIET model achieved optimal results at the third iteration for both the 55 million and 340 million parameter scales.

A.4 Iteration Influence of 740M IIET

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PiQA acc_norm ↑	Hella. acc_norm ↑	SCIQ acc ↑	ARC-c acc_norm ↑	Wino. acc ↑	Avg. ↑
Transformer++	28.2	78.3	28.9	64.3	34.2	76.0	23.6	51.9	46.5
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**
IIET-iter0	27.07	48.52	32.43	65.07	34.80	78.30	23.46	50.36	47.40
IIET-iter1	25.96	36.34	34.43	64.69	36.07	76.30	23.29	50.12	47.48
IIET-iter2	25.49	35.76	34.64	64.96	36.80	77.20	24.23	51.85	48.28

Table 3: Performance comparison of IIET with varying iteration counts at 340 million parameters.

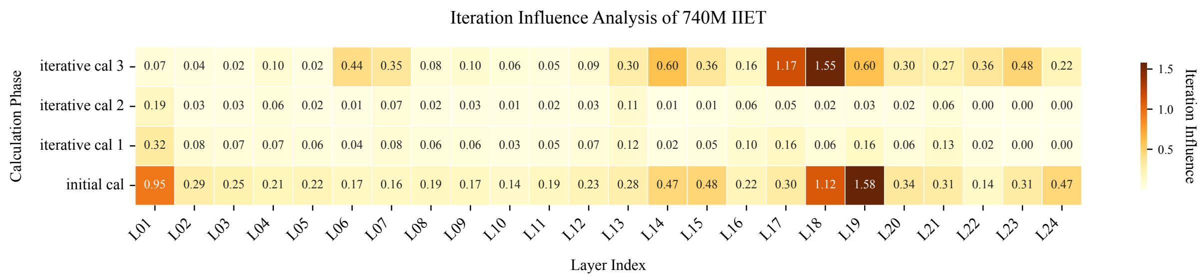

Figure 6: Impact of different iteration stages on the hidden state within each layer of the 740M IIET model, which we term **iteration influence**. Deeper colors indicate larger hidden state changes after this iteration.