## 2. The ODE Perspective on Transformers

In the previous section, we saw that while ODEs were originally designed to model continuous-time processes, they are capable of dealing with discrete-time problems via discretization. In fact, depending on the discretization scheme, we can apply ODEs to different problems. In this section, we discuss one important case where ODEs are fully discretized to inspire neural network design. Specifically, we present interpretations of and improvements to residual network-based models, such as Transformers, from the discretized ODE perspective. We show that ODEs can provide insights into designing such discrete structures.

### 2.1 The Connection Between Transformers and ODEs

Here we consider Transformers as sequence models that take a sequence of tokens as input and produce contextualized representations as output [Vaswani et al., 2017]. For example, most recent **large language models** (LLMs) are based on the Transformer decoder architecture [Brown et al., 2020; Bai et al., 2023; Liu et al., 2024]. This architecture consists of a stack of identical layers, called Transformer layers. Within each Transformer layer, there are two sub-layers: a **multi-head self-attention** (MHSA) sub-layer and a position-wise **feed-forward network** (FFN) sub-layer. A crucial design choice in Transformers is the use of residual connections [He et al., 2016], which introduce a shortcut path adding the input directly to the output of the sub-layers.

In this work, we consider the **pre-norm** architecture, which applies layer normalization to the inputs of the sub-layers rather than after the residual sum [Wang et al., 2019; Baevski and Auli, 2019]. This architecture is widely used in the latest LLMs as it facilitates the training of very deep networks. Formally, let us view the network as a sequence of residual blocks (i.e., sub-layers). Let $\mathbf{x}_l$ denote the input of the $l$-th residual block. The update rule for this block can be expressed as

$$\mathbf{x}_{l+1} \quad = \quad \mathbf{x}_l + F(\mathrm{LN}(\mathbf{x}_l), \theta_l), \tag{1}$$

where $\mathbf{x}_l \in \mathbb{R}^d$ denotes the input, $\mathbf{x}_{l+1} \in \mathbb{R}^d$ denotes the output, $\mathrm{LN}(\cdot)$ denotes the layer normalization function, $F(\cdot)$ denotes the transformation function of the sub-layer (either self-attention or FFN), and $\theta_l$ denotes the parameters of the function $F(\cdot)$.

Here we absorb $\mathrm{LN}(\cdot)$ into $F(\cdot)$, thereby making $F(\cdot)$ a composite function that performs layer normalization followed by the sub-layer transformation. Then, we can rewrite Eq. (1) as

$$\mathbf{x}_{l+1} \quad = \quad \mathbf{x}_l + F(\mathbf{x}_l, \theta_l). \tag{2}$$

This formulation closely resembles the Euler method as discussed in the previous section. To connect Transformer residual blocks to ODEs, let us consider an ODE that describes how a state $\mathbf{x}(t)$ changes over time $t$

$$\frac{d\mathbf{x}(t)}{dt} \quad = \quad f(\mathbf{x}(t), \theta(t), t) \tag{3}$$

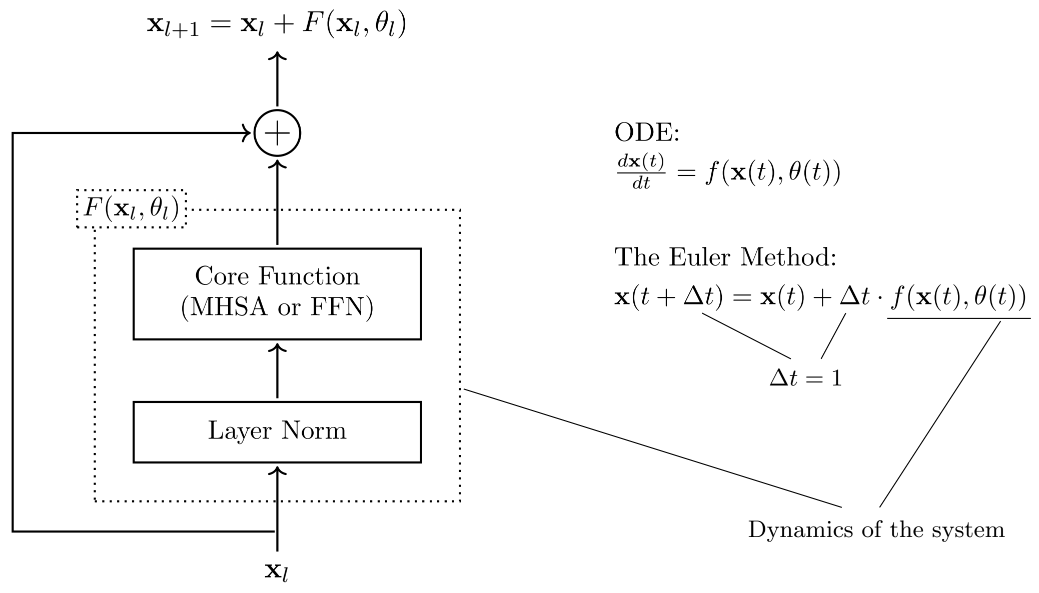

$\mathbf{x}_{l+1} = \mathbf{x}_l + F(\mathbf{x}_l, \theta_l)$
$F(\mathbf{x}_l, \theta_l)$
Core Function (MHSA or FFN)
Layer Norm
$\mathbf{x}_l$
ODE:
$\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), \theta(t))$
The Euler Method:
$\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t), \theta(t))$
$\Delta t = 1$
Dynamics of the system

Figure 1: Illustration of a Transformer sub-layer and the corresponding ODE. The sub-layer can be described as $\mathbf{x}_{l+1} = \mathbf{x}_l + F(\text{LN}(\mathbf{x}_l), \theta_l)$, where the addition of $\mathbf{x}_l$ represents the residual connection and $F(\text{LN}(\mathbf{x}_l), \theta_l)$ represents the transformation defined by the sub-layer. This corresponds to the Euler discretization of the ODE $\frac{d\mathbf{x}(t)}{dt} = f(\mathbf{x}(t), \theta(t))$. More specifically, for the update rule $\mathbf{x}(t+\Delta t) = \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t), \theta(t))$, we treat $f(\mathbf{x}(t), \theta(t))$ as the transformation function of the sub-layer, and set $\Delta t = 1$.

This is a **non-autonomous** ODE, as the dynamics depend explicitly on time through the time-varying parameters $\theta(t)$. We will use $f(\mathbf{x}(t), \theta(t))$ as a shorthand for $f(\mathbf{x}(t), \theta(t), t)$ to keep the notation uncluttered.

Given a discrete time step size $\Delta t$, the state at the next time step can be approximated by the Euler discretization

$$\mathbf{x}(t+\Delta t) \quad = \quad \mathbf{x}(t) + \Delta t \cdot f(\mathbf{x}(t), \theta(t)). \tag{4}$$

By comparing Eq. (2) and Eq. (4), we can establish a direct correspondence between the residual connections in Transformers and the Euler discretization of the ODE. Specifically, if we set the time step size to $\Delta t = 1$, the update rule of the Transformer sub-layer becomes mathematically equivalent to a single step of the Euler method.

In this analogy, the layer index $l$ corresponds to the discrete time step $t$, and the hidden state $\mathbf{x}_l$ corresponds to the state $\mathbf{x}(t)$. As a result, the transformation function $F(\mathbf{x}_l, \theta_l)$ acts as the derivative function $f(\mathbf{x}(t), \theta(t))$ that governs the dynamics in the ODE. It is important to note that while $\theta(t)$ in the continuous ODE formulation represents a parameter trajectory defined continuously over time $t$, the distinct parameters $\theta_l$ in the Transformer can be viewed as the values of this continuous trajectory discretized at integer time steps. Figure 1 illustrates the correspondence between a Transformer sub-layer and the ODE.

Thus, the forward pass of a Transformer with $L$ layers (i.e., $2L$ sub-layers) can be interpreted as numerically integrating an ODE from an initial time $t = 0$ to a final time $t = 2L$ using the Euler method with a unit step size. This perspective suggests that training a Transformer is essentially learning the trajectory of a dynamical system. In this model, the depth of the network is not merely a structural hyperparameter, but can be viewed as the duration of a dynamic process governing the transformation of input representations. Such an insight also motivates the development of neural models based on continuous-time ODEs, such as neural ODEs, which will be discussed in Section 3.

### 2.2 ODE-inspired Architecture Desgin

Although standard pre-norm Transformers can be interpreted as the Euler discretization of ODEs, they have limitations from the perspective of numerical analysis. For example, the Euler discretization is a first-order method and has a local truncation error of $O((\Delta t)^2)$. In Transformers, the error can accumulate significantly as the network grows deeper. As a result, the trajectory computed by the Transformer may deviate from the ideal dynamics of the underlying ODE. Also, the standard Euler method is an explicit method, which is typically considered less numerically stable than implicit ODE solvers. Furthermore, Transformers update representations by executing self-attention and feed-forward layers sequentially, which introduces splitting errors in modeling the coupled dynamics. Therefore, several studies have proposed replacing the simple Euler discretization with more sophisticated schemes to design better neural architectures.

#### 2.2.1 Architectures Interpreted from Physical Systems

One approach is to consider alternative frameworks used in other fields where ODEs have well-established foundations. For example, Lu et al. [2020] interpret Transformers as solvers for the convection-diffusion equation in multi-particle dynamic systems. To be more specific, let us consider a Transformer layer consisting of an MHSA sub-layer and an FFN sub-layer, given by

$$\begin{aligned} \mathbf{x}_{l+1} &= \mathbf{x}_l + F_{\text{att}}(\mathbf{x}_l, \omega_l) && (5) \\ \mathbf{x}_{l+2} &= \mathbf{x}_{l+1} + F_{\text{ffn}}(\mathbf{x}_{l+1}, \pi_{l+1}) && (6) \end{aligned}$$

where $F_{\text{att}}(\cdot)$ and $F_{\text{ffn}}(\cdot)$ denote the functions of the MHSA and FFN sub-layers respectively, and $\omega_l$ and $\pi_{l+1}$ are the corresponding parameters. By substituting Eq. (5) into Eq. (6) and applying a layer index $k$ (letting $\mathbf{h}_k = \mathbf{x}_{2k}$), we obtain the unified expression for the $k$-th Transformer layer:

$$\mathbf{h}_{k+1} = \mathbf{h}_k + F_{\text{att}}(\mathbf{h}_k, \omega_k) + F_{\text{ffn}}\big(\mathbf{h}_k + F_{\text{att}}(\mathbf{h}_k, \omega_k), \pi_k\big) \quad (7)$$

From the perspective of physical systems, this formulation is analogous to the convection–diffusion equation, which models a diffusion process and a convection process simultaneously. Let us define the continuous operators $\text{Att}(\mathbf{h}) = F_{\text{att}}(\mathbf{h}, \omega_k)$ and $\text{FFN}(\mathbf{h}) = F_{\text{ffn}}(\mathbf{h}, \pi_k)$. We can view Eq. (7) as a numerical discretization of the following ODE:

$$\frac{d\mathbf{h}(t)}{dt} = \text{Att}(\mathbf{h}(t)) + \text{FFN}(\mathbf{h}(t)) \quad (8)$$

This ODE differs from those presented earlier in this paper because it contains two distinct terms on the right-hand side. In physics and mathematics, this type of ODE is common, e.g., a physical system evolves under two different forces simultaneously. Solving this combined equation directly is often difficult or computationally expensive. In this case, we often resort to numerical techniques called **operator splitting schemes**, which allow us to approximate the solution by solving the dynamics of each term separately and alternately. From this viewpoint, the sequential update rule of the standard Transformer (Eq. (7)) inherently corresponds to the **Lie-Trotter splitting scheme**, implemented via Euler discretization. In this method, we first advance the state using the diffusion dynamics

$$\frac{d\mathbf{h}(t)}{dt} = \mathrm{Att}(\mathbf{h}(t)) \tag{9}$$

Applying a single Euler step to the initial state $\mathbf{h}_k$ yields the intermediate state $\mathbf{h}'_k = \mathbf{h}_k + \mathrm{Att}(\mathbf{h}_k)$. This is then used as the initial condition for the convection dynamics

$$\frac{d\mathbf{h}(t)}{dt} = \mathrm{FFN}(\mathbf{h}(t)) \tag{10}$$

Similarly, applying a second Euler step yields $\mathbf{h}_{k+1} = \mathbf{h}'_k + \mathrm{FFN}(\mathbf{h}'_k)$. Substituting $\mathbf{h}'_k$ back recovers the exact form of the standard Transformer.

However, the Lie-Trotter splitting is asymmetric (i.e., the order matters), and is only a first-order approximation. To achieve higher accuracy and stability, one can employ the **Strang splitting scheme** (also known as symmetric splitting), which provides a second-order approximation [Strang, 1968]. To be more specific, the Strang splitting scheme uses a symmetric arrangement of the operators. Instead of simply applying $\mathrm{Att}(\cdot)$ followed by $\mathrm{FFN}(\cdot)$, it performs a half-step of one operator, a full step of the other, and finally another half-step of the first operator. Mathematically, if we choose to split the FFN operator, the update rule for a Transformer layer corresponds to the following sequence of compositions

$$\mathbf{h}_{k+1/3} = \mathbf{h}_k + \frac{1}{2}F_{\mathrm{ffn}}(\mathbf{h}_k, \pi_k^{(1)}) \tag{11}$$

$$\mathbf{h}_{k+2/3} = \mathbf{h}_{k+1/3} + F_{\mathrm{att}}(\mathbf{h}_{k+1/3}, \omega_k) \tag{12}$$

$$\mathbf{h}_{k+1} = \mathbf{h}_{k+2/3} + \frac{1}{2}F_{\mathrm{ffn}}(\mathbf{h}_{k+2/3}, \pi_k^{(2)}) \tag{13}$$

This implies a sandwiched structure where one MHSA sub-layer is placed between two FFN sub-layers. The superscripts (1) and (2) indicate that while mathematical Strang splitting typically reuses the same operator, in a neural network context, we can parameterize the two FFN sub-layers with distinct parameters $\pi_k^{(1)}$ and $\pi_k^{(2)}$ to increase expressivity. Theoretically, however, this structure guarantees an improvement in the local truncation error to $O((\Delta t)^3)$ only when the parameters are shared (i.e., $\pi_k^{(1)} = \pi_k^{(2)}$).

The connection between operator splitting schemes and Transformer architectures provides a powerful tool for architecture design: rather than relying solely on trial-and-error, we can derive novel and potentially superior network structures by applying advanced numerical discretization and splitting techniques to the underlying continuous dynamics. For instance, other higher-order splitting methods or adaptive splitting schemes could theoretically motivate further variations of Transformers.

### 2.2.2 Architectures based on Higher-order Methods

The Euler method is a first-order method. A natural improvement is to consider higher-order numerical solvers, which use more sophisticated update rules to achieve higher precision.

A widely-used family of high-order solvers is the Runge-Kutta (RK) series. While the Euler method estimates the solution using only the derivative at the current state, RK methods improve approximation by aggregating derivatives calculated at multiple intermediate points. For instance, in Transformers, the standard residual block can be replaced with a block modeled by RK methods, as proposed by Li et al. [2022]. A general form of the explicit $n$-th order RK method (where $n \leq 4$) with a step size of 1 is given by[1]

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \sum_{i=1}^{n} \gamma_i F_i \tag{14}$$

$$F_1 = F(\mathbf{x}_k, \theta_k) \tag{15}$$

$$F_i = F\left(\mathbf{x}_k + \sum_{j=1}^{i-1} \beta_{ij} F_j, \theta_k\right) \tag{16}$$

where $\gamma_i, \beta_{ij}$ are the coefficients determined by the specific RK scheme (e.g., the classical RK4). In this architecture, each $F_i$ acts as an intermediate estimation of the slope, allowing the network to probe the function landscape at multiple points before making the final update. Theoretically, this RK method achieves a local truncation error of $O((\Delta t)^{n+1})$. Note that the parameters $\theta_k$ are shared within the block. Although we need to evaluate the function $F(\cdot)$ $n$ times, it reuses the same parameters $\theta_k$ for each evaluation. This means that a high-order RK method increases the computational cost to achieve higher precision, but it is parameter efficient.

As an example, an RK2-based Transformer sub-layer can be formulated as

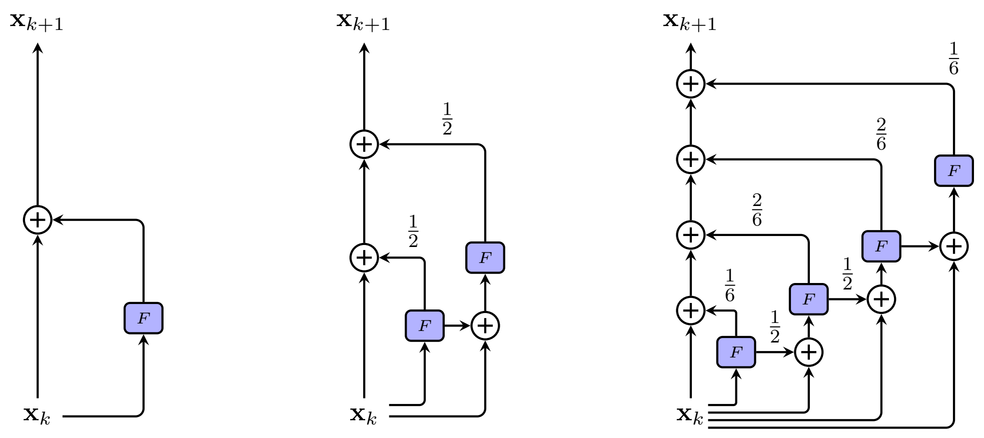
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{1}{2}(F_1 + F_2) \tag{17}$$

$$F_1 = F(\mathbf{x}_k, \theta_k) \tag{18}$$

$$F_2 = F(\mathbf{x}_k + F_1, \theta_k) \tag{19}$$

In this configuration, $F_1$ serves as a predictor, that provides an initial guess of the slope, while $F_2$ acts as a "refiner", that adjusts the update based on the estimated future state. Figure 2 shows illustrations of RK methods with different orders.

1. Strictly speaking, the summation limit $n$ in the equation represents the number of *stages* (function evaluations), while the term *order* refers to the convergence rate of the truncation error. For explicit RK methods, the number of stages equals the order only when the order is $\leq 4$. Achieving an order $p > 4$ requires strictly more than $p$ stages (e.g., a 5th-order method requires at least 6 stages). In this context, we assume $n \leq 4$ for simplicity.



(a) Pre-norm (b) RK2 (c) RK4

Figure 2: Comparison of Transformer sub-layer architectures inspired by numerical ODE solvers [Li et al., 2022]. Subfigure (a): The baseline pre-norm architecture represents a single-step update based on the Euler method. Subfigures (b) and (c): The RK2 and RK4 architectures use multiple function evaluations (stages) within a single block. The output is computed as a weighted aggregation of these intermediate states, as indicated by the coefficients on the edges.

Note that these RK-based architectures introduce a new dimension for scaling neural networks. While current LLM scaling primarily focuses on increasing sequence length [Guo et al., 2025] or inter-layer depth [Yang et al., 2024], RK schemes enable intra-layer depth scaling. This approach provides a parameter-efficient mechanism to decouple computational budget from both parameter count and sequence length.

### 2.2.3 Architectures based on Implicit Methods

Both the Euler method and the standard Runge-Kutta methods described above belong to the category of **explicit methods**, where the next state $\mathbf{x}_{k+1}$ is computed directly from the current state $\mathbf{x}_k$. However, explicit methods can sometimes lack numerical stability. To further improve the model's robustness, one approach is to adopt **implicit methods**, where the next state is defined by an equation involving the unknown future state itself.

Designing architectures based on implicit methods is challenging because solving for $\mathbf{x}_{k+1}$ requires complex root-finding operations. Therefore, more advanced methods are generally used. One example is the PCformer [Li et al., 2024], which adopts a predictor-corrector paradigm with a high-order explicit predictor (like RK) and a multistep implicit corrector.

In the prediction phase, instead of a simple Euler update, PCformer uses a high-order explicit solver (such as the 4th-order RK method) to obtain a high-quality initial estimate, denoted as $\hat{\mathbf{x}}_{k+1}$. For instance, we can use Eqs. (17-19) to get a 2nd-order RK estimate. As an improvement, PCformer uses an **exponential moving average** (EMA) for coefficient learning (i.e., determining $\gamma_i$ in Eq. (14)). In this method, it is hypothesized that higher-order intermediate approximations (e.g., $F_4$ in RK4) are more accurate than lower-order ones (e.g., $F_1$) and thus should contribute more to the output. Instead of using the fixed coefficients prescribed by standard numerical methods (e.g., $1/6, 2/6, 2/6, 1/6$ for RK4), PCformer assigns weights using an EMA decay factor $b$. For an $n$-order predictor, the output is given by

$$\hat{\mathbf{x}}_{k+1} \quad = \quad \mathbf{x}_k + \sum_{i=1}^{n} b(1-b)^{n-i} F_i, \tag{20}$$

where $b$ is a learnable parameter (initialized to 0.5).

In the correction phase, the model employs an implicit linear multistep method (e.g., Adams-Moulton). Since implicit methods require the value of the function at the next time step $F(\mathbf{x}_{k+1})$, which is unknown, the predictor's output $\hat{\mathbf{x}}_{k+1}$ is used as a substitute. The final update rule involves a weighted combination of current evaluations and historical states from previous layers

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha F(\hat{\mathbf{x}}_{k+1}, \theta_k) + \sum_{j=0}^{m} \beta_j F(\mathbf{x}_{k-j}, \theta_{k-j}), \tag{21}$$

where $\alpha$ and $\beta_j$ are learnable coefficients, and $m$ is the number of previous layers the model remembers to help correct the current state. The equation is a learnable Adams-Moulton corrector, where the classical fixed coefficients are replaced by learnable weights $\alpha$ and $\beta_j$ to better adapt to the data distribution. This predictor-corrector architecture reduces truncation errors and enhances generation quality. However, it introduces more computational steps, thus potentially hindering inference efficiency compared to standard Transformers.

To improve inference efficiency, we can adopt the iterative **implicit Euler Transformer** (IIET) architecture [Liu et al., 2025]. Unlike explicit methods that use only the current state to compute the next state, the implicit Euler method is defined as $\mathbf{x}_{k+1} = \mathbf{x}_k + F(\mathbf{x}_{k+1})$. However, solving this equation directly is difficult because $\mathbf{x}_{k+1}$ appears on both sides. IIET addresses this by using fixed-point iteration as a solver within each layer.

Specifically, we start with an initial estimation $\mathbf{x}_{k+1}^{(0)}$ (typically obtained via an explicit Euler step). This estimate is then progressively refined through $r$ iterations:

$$\mathbf{x}_{k+1}^{(0)} \quad = \quad \mathbf{x}_k + F(\mathbf{x}_k, \theta_k) \tag{22}$$

$$\mathbf{x}_{k+1}^{(i)} \quad = \quad \mathbf{x}_k + F(\mathbf{x}_{k+1}^{(i-1)}, \theta_k), \quad \text{for } i = 1, \ldots, r. \tag{23}$$

The final output of the layer is $\mathbf{x}_{k+1} = \mathbf{x}_{k+1}^{(r)}$. This iterative refinement can approach the true solution of the implicit equation more closely than a single-step method. To reduce the inference cost further, one can consider dynamically pruning the number of iterations $r$ for different layers based on their contribution to the final output.

## 2.3 Remarks on Stability and Stiffness

While ODEs offer theoretical interpretations in model design, the stability of the underlying dynamic system remains a challenge when applying these concepts to deep neural networks.

From the perspective of ODEs, the stability of a Transformer is closely related to the smoothness of the information flow and the stiffness of the ODE.

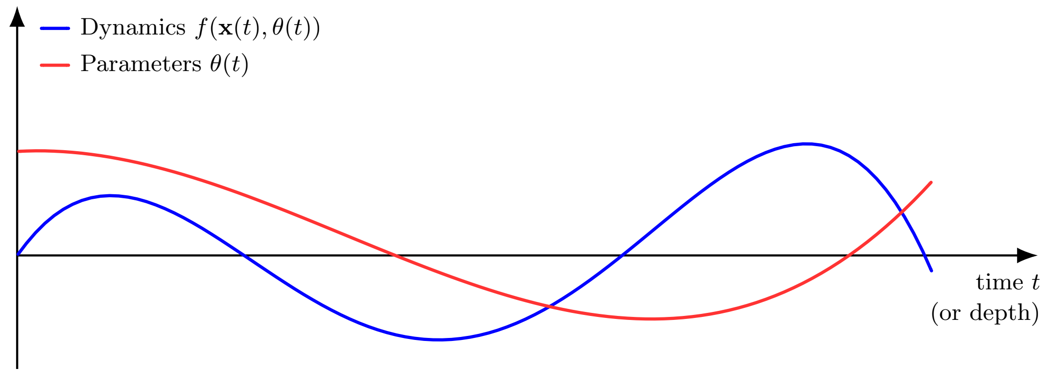
An issue in modeling Transformers as ODEs is that the learned dynamics can be stiff. Here **stiffness** refers to the phenomenon where the continuous dynamics of the hidden states evolve at vastly different rates, e.g., some components change rapidly while others vary slowly. As a result, a stiff ODE is an ODE for which standard explicit numerical methods are severely unstable, thus requiring very small step sizes to solve the equation stably. In the context of neural networks, this implies that standard explicit solvers (or fixed-depth residual connections) may fail to propagate signals effectively, as they are prone to numerical instability when attempting to capture these rapid transient changes without a sufficiently fine-grained discretization. As noted in classical numerical analysis literature [Hairer and Wanner, 1996], explicit methods like the standard Euler or Runge-Kutta methods have a limited region of stability. When applied to stiff problems, they generally require a small step size to avoid numerical divergence. By contrast, implicit methods (e.g., the methods presented in 2.2.3) are often preferred for stiff problems because they possess a much larger stability region.

For Transformers, this issue is more severe due to the discrete nature of the layers. In these models, the parameters $\theta$ are independent and can differ drastically between layers. This contradicts the assumption of smooth dynamics required for stable ODE solutions. Recall the ODE formulation $\frac{d\mathbf{x}(t)}{dt} = F(\mathbf{x}(t), \theta(t))$, where $\theta(t)$ represents parameters that vary continuously with time $t$. However, in standard Transformers, parameters are fixed for each layer, thereby discretizing $\theta(t)$ into a step function (i.e., $\theta_1, \theta_2, \dots$). These abrupt changes in parameters cause the system dynamics to shift sharply from one layer to the next. For instance, in LLMs, it is often observed that the first few layers possess markedly different parameter values, resulting in intermediate representations that differ significantly across layers. Figure 3 shows an illustration of this problem.

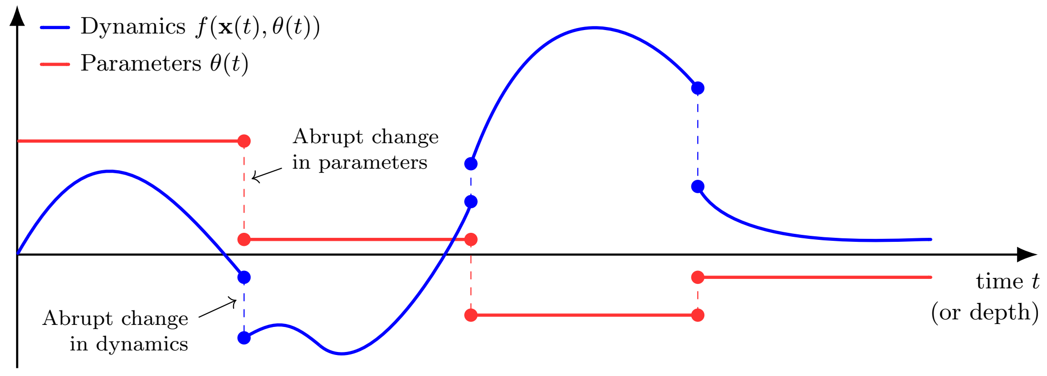
To mitigate the numerical instability caused by stiffness and abrupt parameter shifts, it is natural to constrain the variability of the function $F$. One common way to achieve this is through the Lipschitz continuity condition. Specifically, to ensure a well-conditioned ODE that can be solved stably, there must exist a constant $K$ such that $\|F(\mathbf{x}_1) - F(\mathbf{x}_2)\| \leq K\|\mathbf{x}_1 - \mathbf{x}_2\|$. In neural networks, the Lipschitz constant is closely related to the spectral norm of the weight matrices in each layer [Neyshabur, 2017; Miyato et al., 2018]. The spectral norm is defined as the largest singular value of the matrix. Gouk et al. [2021] demonstrate that constraining the spectral norm can effectively bound the Lipschitz constant. This constraint not only smoothens the dynamics but also ensures that the amplification of signals through the depth of the network remains controlled. This helps prevent numerical divergence.

More broadly, we can achieve higher stability using explicit regularization techniques. Two common strategies can be considered:

- **Parameter Regularization**. To enforce a smoother evolution of the dynamics with respect to depth in Transformers, one can introduce a penalty to the loss function. The following



(a) Smooth evolution of system dynamics and parameters over time



(b) Sharp transitions in system dynamics caused by abrupt changes in parameters

Figure 3: Illustration of the shift in system dynamics caused by abrupt parameter changes. The blue line represents the evolution of system dynamics $f(\mathbf{x}(t), \theta(t))$, and the red line represents the evolution of parameters $\theta(t)$. Subfigure (a) shows an ideal scenario where both parameters and dynamics evolve smoothly and continuously over time. Subfigure (b) shows a realistic scenario where parameters in discrete layers behave as a step function. The abrupt discontinuities in parameters at specific time steps lead to sharp transitions in the system dynamics.

equation shows an example of the penalty term [Haber and Ruthotto, 2017]

$$\mathcal{L}_{\text{param}} = \lambda_{\text{param}} \sum_{k=1}^{L-1} ||\theta_{k+1} - \theta_k||_2^2, \tag{24}$$

where $\lambda_{\text{param}}$ is the weight of the penalty, and $L$ is the depth of the Transformer layer stack[2]. This term constrains adjacent layers to have similar parameters. In the limiting

2. Here, a Transformer layer comprises two sub-layers. $\theta_k$ denotes the collective parameters of the entire layer.

case (e.g., as $\lambda_{\text{param}} \to \infty$), this constraint forces $\theta_{k+1} \approx \theta_k$, which results in parameter-sharing architectures [Dehghani et al., 2019; Lan et al., 2020]. In such models, $\theta$ becomes constant across layers, and the non-autonomous ODE is transformed into an autonomous system (time-invariant). This effectively treats the depth dimension as the time step of a recurrent neural network. In general, autonomous systems are more stable and parameter-efficient.

- **State Regularization**. This method directly controls the change of output across layers. For instance, we can penalize large divergence between outputs of adjacent layers, like this

$$\mathcal{L}_{\text{state}} = \lambda_{\text{state}} \sum_{k=1}^{L-1} ||\mathbf{x}_{k+1} - \mathbf{x}_k||_2^2, \tag{25}$$

where $\lambda_{\text{state}}$ is the weight of this regularization term. In this sense, normalization methods like layer normalization can be seen as an effective way to regularize the hidden states in Transformers. Another approach involves taking the outputs of previous layers as input to produce a combined output. Such layer combination techniques have been extensively used in training deep neural networks [Huang et al., 2017; Wang et al., 2018]. From a numerical analysis perspective, these architectures can be interpreted as multi-step numerical integrators (e.g., linear multistep methods), where the approximation of the next state relies on a history of previous states rather than solely on the immediately preceding one. This sequential dependency effectively smooths the trajectory of the hidden states $\mathbf{x}(t)$. By aggregating information from multiple previous steps, the propagation of signals can be relatively robust even when the discrete transformations vary rapidly.

## References

Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In International Conference on Learning Representations, 2019.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. Qwen technical report. arXiv preprint arXiv:2309.16609, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya

Sutskever, and Dario Amodei. Language models are few-shot learners. Advances in neural information processing systems, 33:1877–1901, 2020.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In International Conference on Learning Representations, 2019.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J Cree. Regularisation of neural networks by enforcing lipschitz continuity. Machine Learning, 110(2):393–416, 2021.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, Hao Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, Tao Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. Nat., 645(8081):633–638, 2025. doi: 10.1038/S41586-025-09422-Z. URL https://doi.org/10.1038/s41586-025-09422-z.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. Inverse problems, 34(1), 2017.

Ernst Hairer and Gerhard Wanner. Solving Ordinary Differential Equations II. Cambridge university press, 1996.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In International Conference on Learning Representations, 2020.

Bei Li, Quan Du, Tao Zhou, Yi Jing, Shuhan Zhou, Xin Zeng, Tong Xiao, Jingbo Zhu, Xuebo Liu, and Min Zhang. Ode transformer: An ordinary differential equation-inspired model for sequence generation. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 8335–8351, 2022.

Bei Li, Tong Zheng, Rui Wang, Jiahao Liu, Junliang Guo, Xu Tan, Tong Xiao, JingBo Zhu, Jingang Wang, and Xunliang Cai. Predictor-corrector enhanced transformers with exponential moving average coefficient learning. Advances in Neural Information Processing Systems, 37: 20358–20382, 2024.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang

Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437, 2024.

Xinyu Liu, Bei Li, Jiahao Liu, Junhao Ruan, Kechen Jiao, Hongyin Tang, Jingang Wang, Tong Xiao, and Jingbo Zhu. Iiet: Efficient numerical transformer via implicit iterative euler method. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 8955–8969, 2025.

Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-yan Liu. Understanding and improving transformer from a multi-particle dynamic system point of view. In ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations, 2020.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In International Conference on Learning Representations, 2018.

Behnam Neyshabur. Implicit regularization in deep learning. arXiv preprint arXiv:1709.01953, 2017.

Gilbert Strang. On the construction and comparison of difference schemes. SIAM journal on numerical analysis, 5(3):506–517, 1968.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

Qiang Wang, Fuxue Li, Tong Xiao, Yanyang Li, Yinqiao Li, and Jingbo Zhu. Multi-layer representation fusion for neural machine translation. In Proceedings of the 27th international conference on computational linguistics, pages 3015–3026, 2018.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F Wong, and Lidia S Chao. Learning deep transformer models for machine translation. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1810–1822, 2019.

Liu Yang, Kangwook Lee, Robert D Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=HHbRxoDTxE.