مجلة جامعة بني وليد للعلوم الإنسانية والتطبيقية
تصدر عن جامعة بني وليد – ليبيا
Website: https://jhas-bwu.com/index.php/bwjhas/index
المجلد التاسع، العدد الأول 2024

تطبيق المعادلات التفاضلية والتكاملات على بنية نموذج لغة الذكاء الاصطناعي

عادل علي دياب
قسم الرياضيات، كلية العلوم، جامعة بني وليد، ليبيا.
adeldiab@bwu.edu.ly

Application of Differential Equations and Integrals on Structure of AI language model

Adel Ali Diab
Department of Mathematics, Faculty of Science, Bani Waleed University, Libya.

تاريخ الاستلام:12-02-2024 تاريخ القبول: 28-02-2024 تاريخ النشر: 11-03-2024

الملخص:

يمكن تعزيز دراسة نماذج لغة الذكاء الاصطناعي، مثل GPT-4، من خلال تطبيق المعادلات التفاضلية والتكاملات. توفر هذه الأساليب فهمًا شاملاً لسلوك النموذج وديناميكيات التعلم والأداء. من خلال صياغة المعادلات التفاضلية الجزئية المناسبة (PDEs)، والمعادلات التفاضلية العادية (ODEs)، والمعادلات التكاملية، يمكن للباحثين تحديد الأنماط التي تساهم في نقاط القوة والضعف في النموذج. يمكن استخدام هذه المعلومات لتصميم خوارزميات تدريب أكثر كفاءة وتحسين إمكانية تفسير النموذج. تعتبر المعادلات التفاضلية الجزئية (PDEs) حاسمة في نمذجة العلاقات بين العناصر المختلفة ضمن تسلسل في نماذج لغة الذكاء الاصطناعي. فهي تلتقط ديناميكيات النموذج، مما يسمح للباحثين بتحليل كيفية معالجة اللغة وتحديد الأنماط التي قد تساهم في نقاط القوة والضعف فيه. تلعب المعادلات التفاضلية العادية (ODEs) دورًا حاسمًا في فهم ديناميكيات التعلم لنماذج لغة الذكاء الاصطناعي، حيث تصف كيفية تغير معلمات النموذج بمرور الوقت أثناء التدريب. يساعد تحليل استقرار ODEs في تصميم خوارزميات تدريب أكثر كفاءة، مما يؤدي إلى تحسين الأداء وقابلية تفسير أفضل. يمكن استخدام المعادلات التكاملية لتقييم أداء نماذج لغة الذكاء الاصطناعي عن طريق حساب مقاييس الأداء المختلفة، مثل الحيرة أو الدقة أو الخسارة. يمكن أن يساعد تحليل هذه المقاييس في توجيه المزيد من التحسينات، مما يؤدي في النهاية إلى أداء أفضل وفهم أكثر شمولاً لمعالجة اللغة الطبيعية. تعد الطرق العددية والحساب الرمزي والحلول التحليلية ضرورية في حل وتحليل النماذج المصاغة، مما يوفر نظرة ثاقبة للهياكل والآليات الرياضية الأساسية. يمكن لهذه المعرفة أن توجه تصميم نماذج لغوية أكثر كفاءة وقابلة للتفسير، مما يؤدي إلى تحسين الأداء وفهم أفضل لمعالجة اللغة الطبيعية.

الكلمات الدالة: الذكاء الاصطناعي ، التفاضلية ، المعادلات ، **GPT-4** ، التكامل.

Abstract

The study of AI language models, such as GPT-4, can be enhanced by applying differential equations and integrals. These methods provide a comprehensive understanding of the model's behavior, learning dynamics, and performance. By formulating appropriate partial differential equations (PDEs), ordinary differential equations (ODEs), and integral equations, researchers can identify patterns that contribute to the model's strengths and weaknesses. This information can be used to design more efficient training algorithms and improve the interpretability of the model. Partial Differential Equations (PDEs) are crucial in modeling the relationships between different elements within a sequence in AI language models. They capture the dynamics of the model, allowing researchers to analyze how it processes language and identify patterns that may contribute to its strengths and weaknesses. Ordinary Differential Equations (ODEs) play a crucial role in understanding the learning dynamics of AI language models, describing how the model's parameters change over time during training. Analyzing the stability of ODEs helps design more efficient training algorithms, leading to improved performance and better interpretability. Integral equations can be used to evaluate the performance of AI language models by calculating various performance metrics, such as perplexity, accuracy, or loss. Analyzing these metrics can help guide further improvements, ultimately leading to better performance and a more comprehensive understanding of natural language processing. Numerical methods, symbolic computation, and analytical solutions are essential in solving and analyzing formulated models, providing insights into the underlying mathematical structures and mechanisms. This knowledge can guide the design of more efficient and interpretable language models, leading to improved performance and a better understanding of natural language processing.

Keywords: AI – Differential – Equations – GPT-4 – Integral.

1. Introduction

The rapid advancements in Artificial Intelligence (AI) have led to the creation of sophisticated AI language models that have revolutionized the way we interact with machines and computers. These models have demonstrated remarkable capabilities in natural language processing, machine translation, and chatbots, enabling seamless communication between humans and machines. As the field of AI continues to evolve, there is a growing need to understand the underlying mathematical foundations that govern the behavior and performance of AI language models **[1]**.

Differential equations and integrals are fundamental concepts in mathematics, widely used in various disciplines such as physics, engineering, and economics to model and analyze complex systems. These mathematical tools have also found applications in AI, particularly in neural networks and reinforcement learning, where they help optimize model parameters and improve performance **[2]**. By exploring the connection between differential equations, integrals, and AI language models, researchers can gain valuable insights into their structure, learning dynamics, and overall performance, leading to more efficient, accurate, and robust language models.

This paper aims to provide an in-depth exploration of the application of differential equations and integrals to the structure of AI language models. In this comprehensive study, we will establish a solid theoretical framework and outline a step-by-step methodology to investigate the role of these mathematical concepts in understanding the behavior of AI language models. The proposed methodology will involve selecting a well-known language model called **GPT-4 [3]**, identifying relevant mathematical concepts, formulating mathematical models, solving and analyzing the models, and validating the results through experimental setups and simulations.

The insights gained from this research will contribute to the advancement of AI language model understanding, enabling researchers to develop more effective and efficient models. Additionally, this study will provide a foundation for future research directions, such as exploring other AI language models, incorporating additional mathematical concepts, or investigating real-world applications of these models in various domains. In the end, this thorough examination of the connection between integrals, differential equations, and AI language models will encourage the creation of state-of-the-art AI systems that may improve communication between humans and machines and stimulate creativity in a variety of sectors.

2. Literature Review

In recent years, a considerable amount of research has been conducted on AI language models, focusing on their mathematical foundations, architectures, and training methodologies. This section will discuss some of the significant literature in this field, highlighting any existing connections between differential equations, integrals, and AI language models.

2.1. Deep Learning and Neural Networks

One of the most popular AI language models is the Recurrent Neural Network (RNN), which uses feedback connections to process sequential data. RNNs have been extensively used in natural language processing tasks, such as language translation and sentiment analysis. The mathematical foundations of RNNs are based on differential equations, specifically the concept of state transition. In their paper "Long short-term memory," Hochreiter and Schmidhuber (1997) introduced the Long Short-Term Memory (LSTM) model, an advanced variant of RNNs that uses differential equations to address the vanishing gradient problem in traditional RNNs [4].

Neural Networks and Deep Learning form the basis for many AI language models, including Recurrent Neural Networks (RNNs) and their advanced variant, Long Short-Term Memory (LSTM) models. These models are particularly useful for processing sequential data, such as natural language processing tasks like language translation and sentiment analysis **[5]**.

RNNs are a type of neural network that uses feedback connections to process sequential data. They are designed to maintain an internal state, known as the "hidden state," which allows them to remember information from previous time steps and use it to process the current input. This ability to remember past inputs is crucial for understanding and generating sequences, such as sentences in a language **[6]**.

Mathematically, the operation of an RNN can be represented using differential equations, specifically the concept of state transition. The state transition equation describes how the hidden state of the RNN changes over time, given the current input and the previous hidden state **[7]**. This equation is often referred to as the "update rule" or "transition function" of the RNN **[8]**. However, RNNs face a significant challenge known as the "vanishing gradient problem." This problem arises when the gradients during backpropagation become either too large or too small, making it difficult for the network to learn long-term dependencies in the input sequence. This issue is particularly problematic for language processing tasks, where understanding the context of a sentence may require considering information from earlier parts of the sentence **[9]**.

To address this problem, Hochreiter and Schmidhuber (1997) introduced the LSTM model. LSTM is an advanced variant of RNNs that uses differential equations to overcome the vanishing gradient problem. LSTM cells consist of multiple interconnected layers, including input, output, and forget gates, as well as memory cells. These gates and memory cells enable the LSTM to control the flow of information and maintain a more stable internal state, allowing it to learn long-term dependencies more effectively **[10]**. The LSTM update rule can be formulated using differential equations. The gates in the LSTM model control the flow of information into and out of the memory cells, which are updated using the input and previous hidden state. The forget gate determines how much information from the previous memory cell should be discarded, while the input gate decides how much new information should be stored. The output gate controls the flow of information from the memory cells to the current hidden state **[11]**.

2.1.1. Mathematical simulation of the LSTM model

To provide a mathematical simulation of the LSTM model, we will break down the model's components and their corresponding equations by Al-Utaibi et al. (2023) [12]. The researchers focus on a single LSTM cell, as the model consists of multiple interconnected cells, as follows

2.1.1.1. Notation and Initialization

Table 1 presents the notation and initializes the variables:

Table 1 Notation and Initialization of a single cell LSTM model.

Variable	Notation and Initialization
T	time step
$\mathbf{i_t}$	input at time step t
$\mathbf{h_t}$	hidden state at time step t
$\mathbf{c_t}$	cell state at time step t
$\mathbf{f_t}$	forget gate at time step t
$\mathbf{i_t}$	input gate at time step t
$\mathbf{o_t}$	output gate at time step t
$\boldsymbol{\sigma}$	sigmoid activation function
Tanh	hyperbolic tangent activation function

2.1.1.2.Forget Gate (f_t)

The forget gate determines how much information from the previous memory cell **[13]** ($c\{t_1\}$) should be discarded. It is calculated using the sigmoid activation function **[14]**:

$$\boldsymbol{f_t = \sigma(W_f \times [i_t \quad h_{t-1}] + b_f)} \qquad \mathbf{(1)}$$

Where;

- W_f is the weight matrix of the forget gate;
- b_f is the bias vector of the forget gate;
- $[i_t \quad h_{t-1}]$ is the concatenation of the current input and the previous hidden state.

2.1.1.3.Input Gate (i_t)

The input gate decides how much new information should be stored in the memory cell [15]. It is also calculated using the sigmoid activation function[16]:

$$\boldsymbol{i_t = \sigma(W_i \times [i_t \quad h_{t-1}] + b_i)} \qquad \mathbf{(2)}$$

Where; W_i and b_i are the weight matrix and bias vector for the input gate, respectively.

2.1.1.4. Memory Cell Update (c_t)

The updated memory cell combines the forgotten information from the previous cell and the new information selected by the input gate **[17]**. It is computed using the hyperbolic tangent activation function **[18]**:

$$\boldsymbol{c_t = f_t \times \tanh(\{c_{t-1}\}) + i_t \times \tanh(W_c \times [i_t \quad h_{t-1}] + b_c)} \quad \mathbf{(3)}$$

Where: W_c and b_c are the weight matrix and bias vector for the memory cell, respectively.

2.1.1.5. Output Gate (o_t)

The output gate controls the flow of information from the memory cell to the current hidden state. It is calculated using the sigmoid activation function **[19]**:

$$o_t = \sigma(W_o \times [i_t \quad h_{t-1}] + b_o) \qquad (4)$$

Where; W_o and b_o are the weight matrix and bias vector for the output gate, respectively.

2.1.1.6. Hidden State (h_t)

The hidden state is determined by the output gate and the updated memory cell [20]:

$$h_t = o_t \times \tanh(c_t) \qquad (5)$$

2.1.1.7. Final output (y_t)

If the LSTM model is used for regression or classification tasks, the final output y_t can be calculated based on the current hidden state

$$y_t = W_y \times h_t + b_y \qquad (6)$$

In practice, LSTM models consist of multiple interconnected cells, and the output of one cell is typically used as the input for the next cell in the sequence. To obtain the final output for an LSTM model with n cells, we need to consider the output from each cell and how they are combined to produce the overall output. In this case, we assume that the LSTM model is used for a sequence-to-sequence task, where the input sequence is transformed into an output sequence of the same length.

Let's denote the output of the i_{th} cell as $h_{t(i)}$, where i ranges from 1 to n. The output of each cell is generated using the equations provided in the previous response. For the final output, we can use a combination of the hidden states from all cells, such as concatenation or summation. Using a simple summation to combine the outputs from all cells:

$$y_t = \sum_{i=1}^{n} (h_{ti}) \qquad (7)$$

This summation represents the final output at time step t for an LSTM model with n cells. In practice, the final output can be further processed using a linear layer or other suitable transformations, depending on the specific task requirements. For example, in a language translation task, the final output may be passed through a softmax function to generate a probability distribution over the vocabulary of the target language.

2.2. Generative Adversarial Networks (GANs)

GANs are another popular AI language model that uses a two-player game between a generator and a discriminator to learn the data distribution and generate new samples. The training process of GANs involves minimizing and maximizing a cost function, which can be formulated using differential equations. The study "Generative Adversarial Networks" by Goodfellow et al. (2014) [21] presented GANs and discussed the mathematical underpinnings of the cost function. The cost function is based on minimax games, a game theory notion that can be expressed by differential equations.

2.2.1. Overview of GANs

Generative Adversarial Networks (GANs) are indeed a class of deep learning models that have gained significant attention in recent years due to their ability to generate new data samples by learning the underlying data distribution from a given dataset [22].

GANs have demonstrated impressive results in various applications, such as image, video, audio, and text generation in natural language processing. The core components of GANs that enable this capability are the generator and the discriminator:

2.2.1.1 *Generator (G)*

The generator (G) is a key component of GANs that takes a random noise vector z as input and produces a fake sample G (z) that resembles the data distribution of the training set. The generator can be represented as a neural network with parameters θ_G [23]. The purpose of the generator is to learn a mapping function that can transform the noise vector into a sample that is indistinguishable from real data samples [24].

The generator network typically consists of multiple layers, including fully connected layers, convolutional layers, and activation functions. These layers help learn complex patterns and features from the input noise vector, allowing the generator to generate diverse and realistic samples [25].

2.2.1.2. Discriminator (D)

The discriminator (D) is another crucial component of GANs that aims to distinguish between real samples from the training set and the fake samples generated by the generator [26]. The discriminator takes an input sample x or a fake sample G(z) and assigns a probability value between 0 and 1, indicating whether it believes the input is real or fake. The discriminator can also be represented as a neural network with parameters θ_D [23].

The discriminator network is typically designed to be a classifier that predicts the probability of the input being a real sample. It also consists of multiple layers, such as fully connected layers, convolutional layers, and activation functions, to learn the features and patterns that distinguish real samples from fake ones [27].

2.2.1.3. Training process

The training process of GANs involves an iterative game between the generator and the discriminator. In each iteration, the generator produces a fake sample G(z), while the discriminator tries to classify it as either real or fake. The goal of the generator is to generate samples that can fool the discriminator, while the discriminator aims to correctly classify both real and fake samples [24].

The performance of both the generator and the discriminator is evaluated using a cost function, which measures the discrepancy between the generated samples and the real data distribution. The cost function can be formulated as a minimax game, where the generator minimizes the cost function while the discriminator maximizes it **[28]**.

2.2.1.4.Applications of GANs

GANs have been successfully applied to various tasks due to their ability to generate new data samples that resemble the underlying data distribution. Some of the notable applications include **[29–31]**:

1. **Image generation**: GANs have been used to generate high-quality images in various domains, such as faces, animals, and landscapes. They have also been applied to image-to-image translation tasks, like converting daytime images to nighttime or summer to winter scenes.
2. **Video generation**: GANs have been extended to generate realistic videos by modeling the temporal dependencies between frames. This has been applied to tasks like action recognition, video inpainting, and video-to-video translation.
3. **Audio generation**: GANs have been employed to generate realistic speech and music, as well as to perform tasks like audio source separation and audio-to-audio translation.
4. **Text generation**: GANs have been applied to natural language processing tasks such as text generation, text-to-speech conversion, and text style transfer.

2.2.2.Mathematical model of GANs

To provide a mathematical simulation of the Generative Adversarial Networks (GANs) model, we will focus on the key components: the generator (G), the discriminator (D), and their cost function. We will use the Lagrangian formulation and differential equations to represent the optimization process.

2.2.2.1.Generator (G)

The generator takes a random noise vector z from a noise distribution $P_z(z)$ and maps it to a fake sample G(z) that resembles the data distribution $P_{data}(x)$. The generator can be represented as a neural network with parameters θ_G. Then $G(z; \theta_G)$ **[32]**.

2.2.2.2.Discriminator (D)

The discriminator takes an input sample x or a fake sample G(z) and assigns a probability value between 0 and 1, indicating whether it believes the input is real or fake. The discriminator can also be represented as a neural network with parameters θ_D. $D(x; \theta_D)$ and $D(G(z); \theta_D)$ **[33]**.

2.2.2.3.*Cost function*

The cost function V(G, D) is defined as the sum of two terms: the logarithm of the probability assigned by the discriminator to a real sample x, and the logarithm of the probability assigned to a fake sample G(z) generated by the generator as follows **[34]**:

$$V(\boldsymbol{G},\boldsymbol{D}) = \boldsymbol{E}_{\boldsymbol{x}}[\log \boldsymbol{D}(\boldsymbol{x})] + \boldsymbol{E}_{\boldsymbol{z}\sim \boldsymbol{P}_{\boldsymbol{z}}}(\boldsymbol{z})[\log \mathbf{1} - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z}))] \quad (8)$$

2.2.2.4.Lagrangian formulation

To represent the optimization process using differential equations, a Lagrange multiplier λ and the Lagrangian L(G, D, λ) are introduced by**[35]**:

$$\mathbf{L(G, D, \lambda) = V(G, D) - \lambda(||G||^2 - K)} \quad (9)$$

Where; K is a constant that ensures the generator's output remains within a certain range.

2.2.2.5.Gradient equations

The gradient of the Lagrangian with respect to the generator, discriminator, and Lagrange multiplier are driven as follow **[24]**:

$$\partial L/\partial G = 0$$

$$\partial L/\partial D = 0 \quad (10)$$

$$\partial L/\partial \lambda = 0$$

2.2.2.6.Differential equations

The gradient equations can be transformed into a system of differential equations by applying the implicit function theorem as follow:

The differential equation for the generator G: $dG/dt = -\partial L/\partial G$

The differential equation for the discriminator D: $dD/dt = -\partial L/\partial D$

The differential equation for Lagrangian L: $dL/dt = -\frac{\partial L}{\partial G} - \frac{\partial L}{\partial D}$

2.2.2.7.Solving the differential equations

The resulting system of differential equations can be solved numerically to find the optimal generator and discriminator parameters (θ_G, θ_D) that minimize and maximize the cost function, respectively. Some popular numerical methods for solving differential equations include the Gradient descent the Euler method, the Runge–Kutta methods, and the Adam optimizer:

Table 2 provides a numerical example for numerical solutions by the above methods:

Table 2 Numerical Solution for GANs model.

Step	**Gradient descent**	**Euler**	**Runge–Kutta**	**Adam optimizer**
Initialization	Assume the initial generator parameter (θ_G) is 0.5, and the initial discriminator parameter (θ_D) is 0.3, and a learning rate (α) of 0.01 for all methods.			
Cost	$L(G, D) = (D(G(z)) - 0.5)^2 + (D(x) - 0.7)^2$			

function				
Calculation	$\partial L/\partial G = 2 * (D(G(z)) - 0.5) * D'(G(z)) * G'(z)$ $\partial L/\partial D = 2 * (D(G(z)) - 0.5) * D'(G(z)) + 2 * (D(x) - 0.7) * D'(x)$	$\partial L/\partial G = 2 * (D(G(z)) - 0.5) * D'(G(z)) * G'(z)$ $\partial L/\partial D = 2 * (D(G(z)) - 0.5) * D'(G(z)) + 2 * (D(x) - 0.7) * D'(x)$	$\partial L/\partial G = 2 * (D(G(z)) - 0.5) * D'(G(z)) * G'(z($ $\partial L/\partial D = 2 * (D(G(z)) - 0.5) * D'(G(z)) + 2 * (D(x) - 0.7) * D'(x($	$m_G = \beta1 * m_G + (1 - \beta1) * \partial L/\partial G$ $m_D = \beta1 * m_D + (1 - \beta1) * \partial L/\partial D$ $v_G = \beta2 * v_G + (1 - \beta2) * (\partial L/\partial G)^2$ $v_D = \beta2 * v_D + (1 - \beta2) * (\partial L/\partial D)^2$ $\theta_{Gnew} = \theta_G - \alpha * m_G / ($ $\theta_{Dnew} = \theta\_D - \alpha * m\_D / (sqrt(v\_D) + \varepsilon)$
Updates	$\Theta_G = \theta_G - \alpha * \partial L/\partial G$ $\theta_D = \theta_D - \alpha * \partial L/\partial D$	$\theta_{Gnew} = \theta\_G + \alpha * \partial L/\partial$ $\theta_{Dnew} = \theta\_D + \alpha * \partial L/\partial D$	$k_{1G} = \alpha * \partial L/\partial G$ $k_{1D} = \alpha * \partial L/\partial D$ $k_{2G} = \alpha * \partial L/\partial G(\theta_G + 0.5 * k_{1G})$ $k2_D = \alpha * \partial L/\partial D(\theta_D + 0.5 * k_{1D})$ $k_{3G} = \alpha * \partial L/\partial G(\theta_G + 0.5 * k2\_G)$ $k_{3D} = \alpha * \partial L/\partial D(\theta_D + 0.5 * k_{2D})$ $k_{4G} = \alpha * \partial L/\partial G(\theta_G + k_{3G})$ $k_{4D} = \alpha * \partial L/\partial D(\theta_D + k_{3D})$ $\theta_{Gnew} = \theta_G + (k_{1G} + 2 *$	$L/\partial G = 2 * (D(G(z)) - 0.5) * D'(G(z)) * G'(z)$ $\partial L/\partial D = 2 * (D(G(z)) - 0.5) * D'(G(z)) + 2 * (D(x) - 0.7) * D'(x()$

			$k_{2G} + 2 * k_{3G} + k_{4G}) / 6$ $\theta Dnew = \theta_D + (k_{1D} + 2 * k_{2D} + 2 * k_{3D} + k_{4D}) / 6$	
Iterative process	Iteration 1: $\theta\_G = 0.5 - 0.01 * (2 * (0.5 - 0.5) * D'(0.5) * G'(0.5)) = 0.5$ $\theta\_D = 0.3 - 0.01 * (2 * (0.5 - 0.5) * D'(0.5) + 2 * (0.7 - 0.7) * D'(0.7)) = 0.3$ Iteration 2: Assuming $D'(0.5) = 0.6$, $D'(0.7) = 0.4$, $G'(0.5) = 0.7$, we get: $\theta\_G = 0.5 - 0.01 * (2 * (0.5 - 0.5) * 0.6 * 0.7) = 0.5$ $\theta\_D = 0.3 - 0.01 * (2 * (0.5 - 0.5) * 0.6 + 2 * (0.7 - 0.7) * 0.4) = 0.296$	Iteration 1: Assuming $D'(0.5) = 0.6$, $D'(0.7) = 0.4$, $G'(0.5) = 0.7$, we get: $\theta_{Gnew} = 0.5 + 0.01 * (2 * (0.5 - 0.5) * 0.6 * 0.7) = 0.5$ $\theta_{Dnew} = 0.3 + 0.01 * (2 * (0.5 - 0.5) * 0.6 + 2 * (0.7 - 0.7) * 0.4) = 0.296$	Iteration 1: Assuming $D'(0.5) = 0.6$, $D'(0.7) = 0.4$, $G'(0.5) = 0.7$, we get: $k1\_G = 0.01 * (2 * (0.5 - 0.5) * 0.6 * 0.7) = 0$ $k1\_D = 0.01 * (2 * (0.5 - 0.5) * 0.6 + 2 * (0.7 - 0.7) * 0.4) = 0$ All other k values will be 0 as well since the parameter updates are also 0. Therefore, $\theta\_G\_new = \theta\_G = 0.5$, and $\theta\_D\_new = \theta\_D = 0.3$.	Assuming $D'(0.5) = 0.6$, $D'(0.7) = 0.4$, $G'(0.5) = 0.7$, we get: $m_G = 0$ $m_D = 0$ $v_G = 0$ $v_D = 0$ $\theta_{Gnew} = \theta_G - \alpha * m_G / (sqrt(vG) + \varepsilon) = 0.5$ $\theta_{Dnew} = \theta_D - \alpha * m\_D / (sqrt(v_D) + \varepsilon) = 0.3$
Multilayer process	**We can continue this process for the remaining iterations, adjusting the gradients, Euler, Runge–Kutta and Adam optimizer based on the updated generator and discriminator parameters. Note that in practice, the methods and updates would be more complex due to the multi–layer structure of the**			

	generator and discriminator networks.

2.3.Transformers

Indeed, Transformers, initially introduced by Vaswani et al. (2017) [36] in their paper "Attention is All You Need," have revolutionized natural language processing (NLP) and serve as the basis for a number of state-of-the-art artificial intelligence (AI) language models. The model can selectively focus on different parts of the input sequence while it processes it, thanks to the self-attention mechanisms at the heart of the transformer design. The relationship between transformers and differential equations, or integrals, may not be as obvious as it is for recurrent neural networks (RNNs) and generative adversarial networks (GANs), but it can still be established by viewing the self-attention mechanism as an optimization problem that can be resolved with the use of mathematical tools like integrals.

2.3.1.Self-attention mechanism in transformers

The model can calculate a weighted sum of values from the input sequence thanks to the self-attention mechanism in transformers [37]. The weights are based on how relevant each input piece is to the current location. A query-key-value technique that calculates a compatibility score between the query and each key in the input sequence is used to determine this relevance. The attention weights are then calculated by scaling and adding the compatibility scores.

Formally, let x = (x_1, x2, …, x_n) be the input sequence, where each xi is a d-dimensional vector. The self-attention mechanism can be defined as follows:

- Query (Q), Key (K), and Value (V) matrices: Q = WQ * x, K = WK * x, V = WV * x

Where; WQ, WK, and WV are learnable weight matrices.

- Scaled dot-product attention: Attention(Q, K, V) = softmax(Q * K^T /$\sqrt{(dk) * V}$)

Where; dk is the dimensionality of the key vectors.

- The output of the self-attention mechanism for each position i is then computed as a weighted sum of the value vectors: Self-attention(x) = [Attention(Q, K, V)] * V

Where; the square brackets denote the matrix operation applied element-wise across the sequence.

2.3.2.Connection to optimization and integration

Despite the self-attention mechanism's seeming lack of relation to integration or optimization, we can make sense of it by approaching it as an optimization problem. Finding the ideal weights (attention scores) for each input element in light of the query and key-value pairs is the

aim of the self-attention mechanism. An optimization problem can be used to formulate this [38]:

- minimize: $F(w) = -\sum w_i \times Q \times \frac{k_i^T}{\sqrt{d_k}}$ subject to: w ≥ 0, and $\sum w_i = 1$

Where; w = (w_i) is the vector of attention weights, and F(w) represents the compatibility score between the query and the input sequence.

This optimization problem can be solved using various techniques, such as gradient-based methods (e.g., stochastic gradient descent) or convex optimization methods. Once the optimal weights are found, they are used to compute the attention scores and the final output of the self-attention mechanism.

2.3.3. Integration and differential equations

The connection between the self-attention mechanism and integration, or differential equations. is not as direct as in the case of RNNs, which can be seen as a special case of ordinary differential equations (ODEs). However, we can still establish a connection by considering the self-attention mechanism as an approximation of a weighted sum of integrals of the input sequence with respect to the attention weights.

Let $f_i(t)$ be the integral of x(t) with respect to t, starting from some initial time t0 and ending at the time corresponding to the its position in the input sequence. Then, the output of the self-attention mechanism can be approximated as a weighted sum of these integrals **[39]**:

Self-attention(x) $\approx \sum(\boldsymbol{w_i} \times \boldsymbol{f_i(t)})$

Where; the attention weights will play a role similar to that of the integration variables. This connection highlights that the self-attention mechanism can be seen as a way to efficiently approximate the integrals of the input sequence, which might be useful in various applications beyond NLP.

2.3.4. Mathematical Model of Transformers

Based on the above, the mathematical model of the transformer architecture, introduced by Vaswani et al. (2017), can be described in terms of its main components: multi-head self-attention, position-wise feed-forward networks (FFNs), and residual connections. We will present the mathematical formulation for each of these components and then combine them to describe the overall transformer architecture.

2.3.4.1. Multi-head self-attention

The multi-head self-attention mechanism allows the model to attend to different parts of the input sequence in parallel. It is composed of multiple attention heads, each focusing on a

particular aspect of the input sequence. The output of each attention head is concatenated and linearly transformed to produce the final output.

Let $x = (x_1, x_2, \ldots, x_n)$ be the input sequence, where each xi is a d-dimensional vector. The multi-head self-attention mechanism can be defined as follows:

- Query (Q), Key (K), and Value (V) matrices for each attention head h: $Q_h = WQ_h * x$, $K_h = WK_h * x$, $V_h = WV_h * x$

Where; WQh, WKh, and WVh are learnable weight matrices for each attention head h.

- Scaled dot-product attention for each attention head h: $Attention\_h(Q_h, K_h, V_h) = softmax(Qh * K_h^T / sqrt(d_k)) * V_h$

Where; d_k is the dimensionality of the key vectors.

- Concatenation and linear transformation of the outputs from all attention heads: $MultiHead(x) = [concAT(Attention_1(Q1, K1, V1), \ldots, Attention\_h(Qh, Kh, Vh))] * WO$

Where; concAT denotes the concatenation of the outputs from all attention heads along the feature dimension, and WO is a learnable weight matrix.

2.3.4.2.Position-wise feed-forward networks (FFNs)

Position-wise feed-forward networks are applied to each position in the input sequence independently. They are composed of a linear transformation and a non-linear activation function, which is typically the ReLU or GeLU function.

Let $x = (x_1, x_2, \ldots, x_n)$ be the input sequence, where each x_i is a d-dimensional vector. The position-wise FFN can be defined as follows:

- Linear transformation: $\mathbf{F(x) = WF * x + b_F}$ **(11)**

Where; WF is a learnable weight matrix, and bF is a learnable bias vector.

- Non-linear activation function (e.g., GeLU or ReLU): $\mathbf{F_{activation}(x) = g(x) = GeLU(x)}$ **or** $\mathbf{g(x) = ReLU(x)}$ **(12)**

2.3.4.3.Residual connections

To improve gradient flow during backpropagation and address the vanishing gradient issue, residual connections are introduced between the input and output of each sub-layer (position-wise FFN or multi-head self-attention).

- Residual connection: $\mathbf{Res(x, y) = x + y}$ **(13)**

2.3.4.4.Overall transformer architecture

The transformer architecture consists of an encoder and a decoder, each composed of multiple identical layers. A position-wise FFN, a residual connection, and a multi-head self-attention mechanism are present in every layer. Furthermore, the relationship between the input

and output sequences is modeled by means of a multi-head attention mechanism between the encoder and decoder.

Let $x_{enc} = (x_{1enc}, x_{2enc}, \ldots, x_{menc})$ be the input sequence to the encoder and $x_{dec} = (x_{1dec}, x_{2dec}, \ldots, x_{ndec})$ be the input sequence to the decoder, where each xi is a d-dimensional vector. The overall transformer architecture can be defined as follows:

- Encoder: For each layer l in the encoder:

$$\mathbf{Encoder_{layerl(xenc)} = Res(MultiHead(encoder_{layer(l-1)}(x_{enc})), encoder_{layer(l-1)}(x_{enc(1-1)}))} \quad (14)$$

- Decoder: For each layer l in the decoder:

$$\mathbf{Decoder_{layer}l(x_{dec}) = Res(MultiHead(decoder_{layer(l-1)}(x_{dec}), encoder_{layers}(x_{enc})), decoder_{layer(l-1)}(x_{dec}))} \quad (15)$$

Where; $encoder_{layers}$ and $decoder_{layers}$ represent the stacked layers of the encoder and decoder, respectively.

2.4. Partial Differential Equations (PDEs) and AI Language Models

The connection between differential equations, integrals, and AI language models can indeed be further explored through the application of partial differential equations (PDEs) in the development of AI language models. PDEs are mathematical equations that describe the behavior of a function with respect to its spatial and temporal variables. They play a crucial role in various scientific disciplines, such as physics, engineering, and computational finance, and have recently gained attention in the context of AI and machine learning.

Alnæs et al. (2014) [40] in their paper "Unified form language: A domain-specific language for weak formulations of partial differential equations" introduced a domain-specific language (DSL) for formulating PDEs. This work demonstrates the potential of using PDEs in the design and analysis of AI language models, as well as other AI applications in scientific computing.

2.4.1. PDEs in AI language models

The application of PDEs in the context of AI language models can be motivated by the need to model and analyze complex relationships between different elements within a given sequence. PDEs can be used to capture these relationships by describing the evolution of a function (e.g., a word embedding or attention score) over time or space [41].

For instance, consider a sequence of words in a language model where each word is associated with a vector representation (embedding). The relationships between these embeddings can be modeled using PDEs to capture the underlying patterns and structures in

the language. This can lead to more accurate and efficient language models, as well as provide insights into the underlying linguistic and semantic properties of the language [42].

2.4.2. The role of the Unified Form Language (UFL)

The Unified Form Language, introduced by Alnæs et al. (2014), is a domain-specific language designed specifically for formulating weak formulations of PDEs [43]. UFL provides a high-level, human-readable syntax for describing PDEs, making it easier for researchers and practitioners to work with PDEs in the context of AI and machine learning applications.

The UFL syntax allows for the definition of PDEs using standard mathematical notation, such as differential operators (e.g., ∇, $\partial/\partial t$), function spaces (e.g., L_2, H_1), and boundary conditions. This makes it possible to formulate PDEs that are tailored to specific AI applications, such as language modeling, image processing, or fluid dynamics simulations [44].

2.4.3. Integration of UFL and AI language models

The integration of UFL and AI language models can be achieved through the following steps [45-47]:

1. **Formulate the PDEs**: The first step is to define the PDEs that describe the desired behavior of the AI language model. This can be done using the UFL syntax, which allows for a concise and accurate representation of the PDEs.
2. Discretize the PDEs: Once the PDEs have been formulated, they need to be discretized to convert them into a form that can be solved numerically using standard machine learning algorithms. This can be done using finite element methods (FEM), finite difference methods (FDM), or other suitable numerical methods.
3. Train the AI language model: With the discretized PDEs in hand, the next step is to train the AI language model using standard machine learning techniques, such as backpropagation and stochastic gradient descent. The PDEs serve as the underlying constraints that guide the learning process, ensuring that the model captures the desired relationships between the input and output sequences.
4. Analyze and interpret the results: Finally, the trained AI language model can be analyzed and interpreted using the insights gained from the PDEs. This can help in understanding the underlying patterns and structures in the language, as well as in identifying potential areas for improvement or further research.

2.4.4. Potential benefits and future directions

The integration of PDEs and UFL in the development of AI language models can lead to several benefits, such as [48-50]:

1) **Improved accuracy and efficiency**: By capturing the underlying relationships between elements in a sequence using PDEs, AI language models can achieve better accuracy and efficiency in processing and understanding natural language.
2) **Enhanced interpretability**: The use of PDEs can provide a more transparent and interpretable understanding of the underlying mechanisms in the language, facilitating the development of more trustworthy AI systems.
3) **Cross-disciplinary collaboration**: The application of PDEs in AI language models can foster collaboration between researchers in mathematics, computer science, and other scientific disciplines, leading to the development of more advanced and impactful AI applications.

3. Theoretical Framework

Differential equations and integrals are fundamental mathematical concepts that have played a crucial role in shaping our understanding of various scientific phenomena and processes. These concepts have been extensively applied in diverse fields, such as physics, engineering, and economics, to model and analyze complex systems. In the following, we will establish a theoretical framework by exploring the role of differential equations and integrals in mathematics, their applications in different fields, and how they can be used to understand the behavior, learning dynamics, and performance of AI language models.

3.1. Role of differential equations and integrals in mathematics

Differential equations are mathematical equations that describe the relationship between a function and its derivatives with respect to one or more independent variables. They are used to model the behavior of systems that change over time or space. Integrals, on the other hand, are mathematical operations that involve the summation or accumulation of values over an interval. They are used to calculate areas, volumes, and other quantities related to the accumulation of values **[51]**.

In mathematics, differential equations and integrals are essential tools for solving problems involving rates of change, accumulation, and optimization. They are used to study the properties of functions, analyze the behavior of dynamical systems, and derive important results in calculus, such as the Fundamental Theorem of Calculus **[52]**.

3.2. Applications in various fields

Differential equations and integrals have found wide applications in various fields, including **[53]**:

1) **Physics**: Differential equations are used to model and analyze physical systems, such as the motion of celestial bodies, the behavior of fluids, and the propagation of electromagnetic waves. Integrals are used to calculate quantities like work, energy, and momentum in physics.
2) **Engineering**: Differential equations are used to design and analyze engineering systems, such as control systems, electrical circuits, and structural mechanics. Integrals are used in areas like fluid dynamics, heat transfer, and signal processing.
3) **Economics**: Differential equations are used to model economic systems, such as growth models, market dynamics, and financial derivatives. Integrals are used in areas like utility theory, cost-benefit analysis, and optimization problems.

3.3. Understanding the behavior, learning dynamics, and performance of AI language model

Differential equations and integrals can be applied to understand the behavior, learning dynamics, and performance of AI language models in the following ways **[54-56]**:

1) **Modeling linguistic phenomena**: Differential equations can be used to model various linguistic phenomena, such as the diffusion of meaning, the spread of linguistic innovations, or the evolution of language structures over time. For instance, a differential equation could describe how the meaning of a word diffuses through a language community, with the rate of diffusion depending on factors such as the word's frequency, context, or social network structure.
2) **Regularization and prior knowledge**: Differential equations can be used to incorporate prior knowledge or regularization into AI language models. By imposing constraints on the model's parameters or outputs through differential equations, we can encourage the model to learn more interpretable or desirable patterns. For example, a differential equation-based regularization term could enforce smoothness in the model's predictions or encourage the model to learn a specific linguistic property, such as grammaticality or semantic coherence.
3) **Efficient inference and optimization**: Differential equations can be used to develop efficient inference and optimization algorithms for AI language models. For instance, the finite element method (FEM) is a widely used numerical method for solving differential equations, which can be applied to optimize the model's parameters or compute the model's predictions. FEM discretizes the problem space into smaller elements, allowing for more efficient computations and parallelization.

4) **Performance evaluation**: Integrals can be used to evaluate the performance of AI language models by calculating metrics like perplexity, accuracy, or loss. For example, the negative log-likelihood of a language model can be computed using integrals to assess its predictive performance on a given dataset.

4.Methodology

To apply differential equations and integrals to the study of AI language models, the researcher follows these specific steps and procedures:

4.1.Selecting an appropriate AI language model

For this demonstration, we will choose the Transformer-based model called GPT-4. GPT-4 is a powerful language model that has been trained on a vast amount of text data to generate human-like text and understand complex language structures. It uses a self-attention mechanism to process input sequences and has a hierarchical architecture that includes an encoder and a decoder [57].

4.2.Identifying Mathematical Concepts

In this work, the researcher considers the following differential equations and integrals for the study of GPT-4:

1) **Partial Differential Equations (PDEs)**: PDEs capture the interactions between various items in a sequence and are used to simulate the evolution of attention scores or embedding in the GPT-4.
2) **Ordinary Differential Equations (ODEs)**: ODEs are employed to model the learning dynamics of GPT-4, describing how the model's parameters change over time during training.
3) **Integral Equations**: Integral equations are utilized to analyze the performance of GPT-4 by computing metrics like perplexity or accuracy.

4.3.Mathematical Models

The mathematical models that describe the behavior of GPT-4 using the chosen differential equations and integrals are:

1) **PDEs for attention scores or embedding**: In order to better grasp the underlying patterns and structures in the language for enhanced performance or interpretability, we create a PDE that represents the evolution of the attention scores or embedding in GPT-4 over time or space.

2) **ODEs for learning dynamics**: We formulate an ODE that describes how the model's parameters change over time during training, providing insights into the learning process of GPT−4 and guiding the design of more efficient training algorithms.
3) **Integral equations for performance evaluation**: We formulate an integral equation that calculates the performance metrics of GPT−4, such as perplexity or accuracy, to understand the model's strengths and weaknesses and guide further improvements.

4.4.Application on Numerical Solution of GPT−4

Let's consider a simple example where the model is a change in attention scores over time. The researcher uses a 1D PDE to represent this:

$$\frac{\partial A(x,t)}{\partial t} = D \times \frac{\partial^2(x,t)}{\partial t^2} + f(A(x,t)) \qquad (16)$$

Where; A(x,t) represents the attention score at position x and time t, D is a diffusion coefficient, and f(A(x,t)) represents an interaction function that captures the influence of neighboring attention scores.

For ODEs for learning dynamics, the researcher formulates an ODE that describes how the model's parameters change over time during training. This provides insights into the learning process of GPT−4 and helps in designing more efficient training algorithms.

As an example, let's consider an ODE that models the change in a single parameter θ during training:

$$\frac{d\theta(t)}{dt} = \eta \times (\theta(t) - \theta_{target}) \qquad (17)$$

Where; $\theta(t)$ represents the parameter value at time t, η is the learning rate, and θ_{target} is the target value for the parameter.

For Integral equations for performance evaluation: the researcher formulates an integral equation that calculates the performance metrics of GPT−4 by perplexity as follow:

$$Perplexity = e^{-\sum i(log(P(x_i)|context)} \qquad (18)$$

Where; $P(x_i \mid context)$ represents the probability of generating the ith word in a sequence, given the context words, and the summation is performed over all words in the sequence.

To solve and analyze the formulated models, the researcher employs the following techniques:

1) **Numerical methods**: For solving the PDEs and ODEs, the researcher uses numerical methods like finite difference methods (FDM), finite element methods (FEM), or spectral methods. These methods discretize the problem space into smaller elements, allowing for more efficient computations and parallelization.

Let's use a numerical example for the PDE mentioned earlier. We can discretize the spatial domain using a grid and approximate the derivatives using finite differences. Then, we can use an iterative method like the time-stepping method to solve the PDE at each time step.

2) **Symbolic computation**: For deriving and manipulating the equations, we can use symbolic computation tools like Mathematica or SymPy. These tools can help in simplifying and solving the equations analytically, providing insights into the underlying mathematical structures.

As an example, the researcher uses symbolic computation to analyze the stability of the ODE for the learning dynamics as presented in equation 17. By analyzing the characteristic equation, we can determine the stability of the system and understand how the parameter θ converges to its target value.

3) **Analytical solutions**: For some simpler cases, the researcher finds analytical solutions to the formulated models. These solutions provide valuable insights into the model's behavior and help in understanding the underlying mechanisms.

For instance, we can find an analytical solution for the ODE describing the learning dynamics if the target value θ_{target} is constant:

$$\boldsymbol{\theta(t) = \theta_0 + \eta \times (\theta_{target} - \theta_0) \times t} \qquad \textbf{(19)}$$

Where; θ_0 is the initial parameter value, and this solution shows that the parameter θ converges linearly to the target value θ target.

5.Discussion and Results

In this section, the researcher delves deeper into the applications of differential equations and integrals in the context of AI language models, such as GPT-4. The discussion shows how these mathematical concepts can provide insights into the model's behavior, learning dynamics, and performance, and how they can guide the design of more efficient and interpretable language models.

5.1.Understanding the Behavior of AI Language Models

Differential equations can be used to model the behavior of AI language models by capturing the relationships between different elements within a sequence. For instance, a PDE could describe how the meaning of a word diffuses through a language community, with the rate of diffusion depending on factors such as the word's frequency, context, or social network structure. By analyzing the PDE, researchers can gain insights into the underlying linguistic phenomena and develop models that better capture these patterns.

5.2. Modeling the Learning Dynamics of AI Language Models

ODEs can be employed to model the learning dynamics of AI language models, describing how the model's parameters change over time during training. For example, an ODE that models the change in a single parameter θ during training can provide insights into the learning process of GPT-4 and help in designing more efficient training algorithms. By analyzing the stability of the ODE, researchers can understand how the parameter θ converges to its target value and use this information to improve the training process.

5.3. Evaluating the Performance of AI Language Models

Integral equations can be used to evaluate the performance of AI language models by calculating metrics like perplexity, accuracy, or loss. For instance, the perplexity of a language model can be computed using integrals to assess its predictive performance on a given dataset. By analyzing the integral equation, researchers can understand the model's strengths and weaknesses and guide further improvements.

5.4. Numerical Methods, Symbolic Computation, and Analytical Solutions

Numerical methods, symbolic computation, and analytical solutions play crucial roles in solving and analyzing the formulated models. Numerical methods like finite difference methods (FDM), finite element methods (FEM), or spectral methods discretize the problem space into smaller elements, allowing for more efficient computations and parallelization. Symbolic computation tools like Mathematica or SymPy can help in simplifying and solving the equations analytically, providing insights into the underlying mathematical structures. Analytical solutions for simpler cases offer valuable insights into the model's behavior and help in understanding the underlying mechanisms.

6. Conclusion

The application of differential equations and integrals to the study of AI language models like GPT-4 can lead to a deeper understanding of the model's behavior, learning dynamics, and performance. By formulating appropriate PDEs, ODEs, and integral equations, and employing numerical methods, symbolic computation, and analytical solutions, researchers can gain valuable insights into the underlying linguistic phenomena, the learning process, and the model's strengths and weaknesses. These insights can guide the design of more efficient and interpretable language models, ultimately leading to improved performance and a better understanding of natural language processing.

References

[1]W. Wang and K. Siau, "Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda," *Journal of Database Management (JDM),* vol. 30, pp. 61-79, 2019.

[2]C. Rackauckas, Y. Ma, J. Martensen, C. Warner, K. Zubov, R. Supekar*, et al.*, "Universal differential equations for scientific machine learning," *arXiv preprint arXiv:2001.04385,* 2020.

[3]C. du Plooy and R. Oosthuizen, "AI usefulness in systems modelling and simulation: gpt-4 application," *South African Journal of Industrial Engineering,* vol. 34, pp. 286-303, 2023.

[4]S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation,* vol. 9, pp. 1735-1780, 1997.

[5]W. C. Wong, E. Chee, J. Li, and X. Wang, "Recurrent neural network-based model predictive control for continuous pharmaceutical manufacturing," *Mathematics,* vol. 6, p. 242, 2018.

[6]A. L. Caterini, D. E. Chang, A. L. Caterini, and D. E. Chang, "Recurrent neural networks," *Deep neural networks in a mathematical framework,* pp. 59-79, 2018.

[7]E. Wigner, "The transition state method," *Transactions of the Faraday Society,* vol. 34, pp. 29-41, 1938.

[8]A. Salaün, Y. Petetin, and F. Desbouvries, "Comparing the modeling powers of RNN and HMM," in *2019 18th ieee international conference on machine learning and applications (icmla)*, 2019, pp. 1496-1499.

[9]S.-H. Noh, "Analysis of gradient vanishing of RNNs and performance comparison," *Information,* vol. 12, p. 442, 2021.

[10]M. A. I. Sunny, M. M. S. Maswood, and A. G. Alharbi, "Deep learning-based stock price prediction using LSTM and bi-directional LSTM model," in *2020 2nd novel intelligent and leading emerging sciences conference (NILES)*, 2020, pp. 87-92.

[11]F. Shahid, A. Zameer, and M. Muneeb, "A novel genetic LSTM model for wind power forecast," *Energy,* vol. 223, p. 120069, 2021.

[12]K. A. Al-Utaibi, S. Siddiq, and S. M. Sait, "Stock Price forecasting with LSTM: A brief analysis of mathematics behind LSTM," *Biophysical Reviews and Letters,* pp. 1-14, 2023.

[13]J. Van Der Westhuizen and J. Lasenby, "The unreasonable effectiveness of the forget gate," *arXiv preprint arXiv:1804.04849,* 2018.

[14]K. Vijayaprabakaran and K. Sathiyamurthy, "Towards activation function search for long short-term model network: A differential evolution based approach," *Journal of King Saud University-Computer and Information Sciences,* vol. 34, pp. 2637-2650, 2022.

[15]K. Yao, T. Cohn, K. Vylomova, K. Duh, and C. Dyer, "Depth-gated LSTM," *arXiv preprint arXiv:1508.03790,* 2015.

[16]A. Pulver and S. Lyu, "LSTM with working memory," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 845-851.

[17]F. Landi, L. Baraldi, M. Cornia, and R. Cucchiara, "Working memory connections for LSTM," *Neural Networks,* vol. 144, pp. 334-341, 2021.

[18]P. Mei, C. Huang, D. Yang, S. Yang, F. Chen, and Q. Song, "A vehicle-cloud collaborative strategy for state of energy estimation based on CNN-LSTM networks," in *2022 2nd International Conference on Computers and Automation (CompAuto)*, 2022, pp. 128-132.

[19]Z. Chang, Y. Zhang, and W. Chen, "Effective adam-optimized LSTM neural network for electricity price forecasting," in *2018 IEEE 9th international conference on software engineering and service science (ICSESS)*, 2018, pp. 245-248.

[20]J. Hwang, "Modeling Financial Time Series using LSTM with Trainable Initial Hidden States," *arXiv preprint arXiv:2007.06848,* 2020.

[21]I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair*, et al.*, "Generative adversarial networks," *Communications of the ACM,* vol. 63, pp. 139-144, 2020.

[22]Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng, "Recent progress on generative adversarial networks (GANs): A survey," *IEEE access,* vol. 7, pp. 36322-36333, 2019.

[23]P. Manisha and S. Gujar, "Generative Adversarial Networks (GANs): What it can generate and What it cannot?," *arXiv preprint arXiv:1804.00140,* 2018.

[24]D. Saxena and J. Cao, "Generative adversarial networks (GANs) challenges, solutions, and future directions," *ACM Computing Surveys (CSUR),* vol. 54, pp. 1-42, 2021.

[25]P. Salehi, A. Chalechale, and M. Taghizadeh, "Generative adversarial networks (GANs): An overview of theoretical model, evaluation metrics, and recent developments," *arXiv preprint arXiv:2005.13178,* 2020.

[26]T. Nguyen, T. Le, H. Vu, and D. Phung, "Dual discriminator generative adversarial nets," *Advances in neural information processing systems,* vol. 30, 2017.

[27]V. Zadorozhnyy, Q. Cheng, and Q. Ye, "Adaptive weighted discriminator for training generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4781-4790.

[28]L. Gonog and Y. Zhou, "A review: generative adversarial networks," in *2019 14th IEEE conference on industrial electronics and applications (ICIEA)*, 2019, pp. 505-510.

[29]H. Alqahtani, M. Kavakli-Thorne, and G. Kumar, "Applications of generative adversarial networks (gans): An updated review," *Archives of Computational Methods in Engineering,* vol. 28, pp. 525-552, 2021.

[30]S. Porkodi, V. Sarada, V. Maik, and K. Gurushankar, "Generic image application using GANs (generative adversarial networks): a review," *Evolving Systems,* vol. 14, pp. 903-917, 2023.

[31]A. Dash, J. Ye, and G. Wang, "A review of Generative Adversarial Networks (GANs) and its applications in a wide variety of disciplines: From Medical to Remote Sensing," *IEEE Access,* 2023.

[32]B. Yilmaz and K. Ralf, "Understanding the mathematical background of Generative Adversarial Networks (GANs)," *Mathematical Modelling and Numerical Simulation with Applications,* vol. 3, pp. 234-255, 2023.

[33]P. Lencastre, M. Gjersdal, L. R. Gorjão, A. Yazidi, and P. G. Lind, "Modern AI versus century-old mathematical models: How far can we go with generative adversarial networks to reproduce stochastic processes?," *Physica D: Nonlinear Phenomena,* vol. 453, p. 133831, 2023.

[34]C. He, S. Huang, R. Cheng, K. C. Tan, and Y. Jin, "Evolutionary multiobjective optimization driven by generative adversarial networks (GANs)," *IEEE transactions on cybernetics,* vol. 51, pp. 3129-3142, 2020.

[35]S. Asokan and C. S. Seelamantula, "Euler-Lagrange Analysis of Generative Adversarial Networks," *Journal of Machine Learning Research,* vol. 24, pp. 1-100, 2023.

[36]A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez*, et al.*, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.

[37]B. Ghojogh and A. Ghodsi, "Attention mechanism, transformers, BERT, and GPT: tutorial and survey," 2020.

[38]J. Saha, D. Hazarika, N. B. Y. Gorla, and S. K. Panda, "Machine-learning-aided optimization framework for design of medium-voltage grid-connected solid-state transformers," *IEEE Journal of Emerging and Selected Topics in Power Electronics,* vol. 9, pp. 6886-6900, 2021.

[39]A. Bryutkin, J. Huang, Z. Deng, G. Yang, C.-B. Schönlieb, and A. Aviles-Rivero, "HAMLET: Graph Transformer Neural Operator for Partial Differential Equations," *arXiv preprint arXiv:2402.03541,* 2024.

[40]M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, "Unified form language: A domain-specific language for weak formulations of partial differential equations," *ACM Transactions on Mathematical Software (TOMS),* vol. 40, pp. 1-37, 2014.

[41]L. Yang, S. Liu, T. Meng, and S. J. Osher, "In-context operator learning with data prompts for differential equation problems," *Proceedings of the National Academy of Sciences,* vol. 120, p. e2310142120, 2023.

[42]K. C. Cheung and S. See, "Recent advance in machine learning for partial differential equation," *CCF Transactions on High Performance Computing,* vol. 3, pp. 298-310, 2021.

[43]A. Clark and A. Evans, "Foundations of the Unified Modeling Language," in *Proceedigs of the 2nd Northern Formal Methods Workshop.*, 1997.

[44]N. Bouziani and D. A. Ham, "Escaping the abstraction: a foreign function interface for the Unified Form Language [UFL]," *arXiv preprint arXiv:2111.00945,* 2021.

[45]X. Yang, A. Chen, N. PourNejatian, H. C. Shin, K. E. Smith, C. Parisien*, et al.*, "Gatortron: A large clinical language model to unlock patient information from unstructured electronic health records," *arXiv preprint arXiv:2203.03540,* 2022.

[46]G. P. Wellawatte and P. Schwaller, "Extracting human interpretable structure-property relationships in chemistry using XAI and large language models," *arXiv preprint arXiv:2311.04047,* 2023.

[47]G. A. Katuka, S. Chakraburty, H. Lee, S. Dhama, T. Earle-Randell, M. Celepkolu*, et al.*, "Integrating Natural Language Processing in Middle School Science Classrooms: An Experience Report," 2024.

[48]M. Magill, "Opportunities for the Deep Neural Network Method of Solving Partial Differential Equations in the Computational Study of Biomolecules Driven Through Periodic Geometries," University of Ontario Institute of Technology, 2022.

[49]N. Koceska, S. Koceski, L. K. Lazarova, M. Miteva, and B. Zlatanovska, "Can ChatGPT be used for solving ordinary differential equations," *Balkan Journal of Applied Mathematics and Informatics,* vol. 6, pp. 103-114, 2023.

[50]A. Pyrkov, A. Aliper, D. Bezrukov, D. Podolskiy, F. Ren, and A. Zhavoronkov, "Complexity of life sciences in quantum and AI era," *Wiley Interdisciplinary Reviews: Computational Molecular Science,* vol. 14, p. e1701, 2024.

[51]P. Agarwal, R. P. Agarwal, and M. Ruzhansky, *Special functions and analysis of differential equations*: CRC Press, 2020.

[52]P. W. Thompson and T. Dreyfus, "A coherent approach to the Fundamental Theorem of Calculus using differentials," in *Proceedings of the Conference on Didactics of Mathematics in Higher Education as a Scientific Discipline*, 2017, pp. 354-358.

[53]S.-B. Hsu and K.-C. Chen, *Ordinary differential equations with applications* vol. 23: World scientific, 2022.

[54]K. Zhang, K. Zhang, M. Zhang, H. Zhao, Q. Liu, W. Wu, *et al.*, "Incorporating dynamic semantics into pre-trained language model for aspect-based sentiment analysis," *arXiv preprint arXiv:2203.16369,* 2022.

[55]J. L. McClelland, F. Hill, M. Rudolph, J. Baldridge, and H. Schütze, "Placing language in an integrated understanding system: Next steps toward human-level performance in neural language models," *Proceedings of the National Academy of Sciences,* vol. 117, pp. 25966-25974, 2020.

[56]L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, *et al.*, "A survey on large language model based autonomous agents," *arXiv preprint arXiv:2308.11432,* 2023.

[57]J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774,* 2023.