APLICAÇÃO DE REDES NEURAIS PARA O PROBLEMA DE TRÊS CORPOS

Vinícius Manuel Martins, Aurélio Faustino Hoppe – Orientador

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

viniciusmanuel@furb.br, aureliof@furb.br

Resumo: *Este estudo aborda o problema dos três corpos utilizando redes neurais para prever as forças gravitacionais aplicadas a cada corpo. A metodologia envolveu o desenvolvimento de modelos baseados em Redes Neurais Artificiais (RNAs), Convolutional Neural Network (CNNs) e Recurrent Neural Network (RNNs), avaliando diferentes arquiteturas e hiperparâmetros. Além disso, para a validação do efeito caótico e identificação de padrões foi desenvolvida uma ferramenta de visualização das simulações. Os resultados indicam que a CNN foi a que melhor generalizou o comportamento caótico do sistema, apresentando uma correlação positiva entre a eficácia do modelo e a capacidade de lidar com a simulação proposta por Chenciner e Montgomery (2000). Conclui-se que as redes neurais se mostraram promissoras para a realização de simulações gravitacionais. Entretanto, desafios como o overfitting e a complexidade inerente à simulação de sistemas com mais de três corpos demandam estudos mais aprofundados.*

Palavras-chave*: Simulação gravitacional. Problema dos três corpos. Comportamento. RNAs. CNNs. RNNs.*

1 INTRODUÇÃO

A dança cósmica dos corpos celestes sempre fascinou a humanidade, desde os primeiros astrônomos que buscavam padrões no céu noturno até os cientistas modernos que desvendam os mistérios do universo (ALVES-BRITO; CORTESI, 2021). Compreender o movimento desses corpos celestes é fundamental para desvendar a estrutura e a evolução do cosmos (ANGULO; WHITE, 2010). No cerne dessa busca por conhecimento reside o problema de três corpos, um desafio clássico da física que intriga pesquisadores há séculos. Esse problema, aparentemente simples em sua formulação, consiste em determinar o movimento de três corpos celestes – sejam eles estrelas, planetas ou satélites – que interagem gravitacionalmente (BREEN *et al.*, 2020). Apesar da aparente simplicidade, a resolução analítica do problema de três corpos, governado pela Lei da Gravitação Universal de Newton, é extremamente complexa devido à natureza não linear das equações diferenciais envolvidas (WANG *et al.*, 2020).

Para Hernandez (2019), a complexidade matemática decorrente da interação mútua entre os três corpos impossibilita encontrar uma solução analítica geral que descreva o movimento do sistema para qualquer instante de tempo. Embora mentes brilhantes como Lagrange tenham se dedicado a encontrar soluções para casos específicos, como o problema restrito de três corpos, a busca por uma solução geral que abrangesse todas as condições iniciais possíveis se mostrou infrutífera (MUSIELAK; QUARLES, 2014).

Dada a impossibilidade de encontrar uma solução analítica geral, a comunidade científica voltou-se para as simulações computacionais como alternativa para estudar o problema de três corpos (GARRISON *et al.*, 2021). Com o advento dos computadores, métodos numéricos emergiram, utilizando o poder de cálculo para aproximar o movimento dos corpos celestes. No entanto, essa abordagem também apresenta desafios significativos. A necessidade de discretizar o tempo – ou seja, dividir o tempo em intervalos fixos para realizar os cálculos – introduz erros numéricos que se acumulam ao longo do tempo, limitando a precisão das previsões a longo prazo. Além disso, erros de arredondamento e truncamento, inerentes à representação finita dos números reais em computadores, também contribuem para a imprecisão dos resultados (MUKHERJEE *et al.*, 2021).

Esses erros numéricos são amplificados quando os corpos se aproximam uns dos outros, levando a divergências e resultados imprecisos (BREEN *et al.*, 2020). O efeito borboleta, um dos pilares da teoria do caos, demonstra que mesmo minúsculas variações nas condições iniciais podem conduzir a trajetórias completamente distintas, tornando a previsão do movimento a longo prazo um desafio formidável. Como destacam Gonzalez *et al.* (2020), essas imprecisões, inerentes às simulações numéricas, podem transformar um sistema que, em teoria, seria determinístico em algo imprevisível a longo prazo.

Diante da complexidade característica do problema de três corpos e das limitações das soluções tradicionais, este estudo investiga o potencial das redes neurais artificiais para modelar e prever o movimento de partículas nesse sistema complexo. Acredita-se que a capacidade das redes neurais de aprender padrões complexos a partir de dados, aliada à sua eficiência computacional em cenários de grande escala, oferece uma nova perspectiva para a solução deste desafio clássico da física e da astrofísica. Portanto, o objetivo principal deste trabalho é avaliar a capacidade de diferentes arquiteturas de redes neurais em prever as trajetórias de três corpos em interação gravitacional a longo prazo, considerando a precisão e

o tempo de processamento. Para isso, serão exploradas Redes Neurais Artificiais (RNAs), Redes Neurais Recorrentes (RNNs) e Redes Neurais Convolucionais (CNNs), sendo o desempenho de cada modelo avaliado utilizando métricas como Erro Quadrático Médio (EQM), tempo de inferência e o comportamento frente ao efeito caótico.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está subdividido em cinco seções. A seção 2.1 abordará a simulação de N Corpos. Na seção 2.2, serão descritas as Redes Neurais, discutindo suas características e aplicações. A seção 2.3 tratará das Redes Neurais Convolucionais, enquanto a seção 2.4 explorará o potencial das Redes Neurais Recorrentes, destacando sua capacidade de aprender padrões complexos e realizar previsões de alta precisão. Por fim, a seção 2.5 apresentará os trabalhos correlatos identificados durante a revisão bibliográfica, contextualizando a pesquisa atual em relação ao estado da arte.

2.1 SIMULAÇÃO DE N CORPOS

A simulação de N corpos é um problema clássico da mecânica celeste, que envolve a predição das posições e velocidades de um grupo astros, como estrelas ou planetas, com base em suas posições e velocidades iniciais, bem como suas interações gravitacionais mútuas (SPURZEM, 1999).

O problema de N corpos é notoriamente difícil de resolver com exatidão, exceto em casos especiais, devido à natureza não linear das equações envolvidas. No entanto, existem várias abordagens numéricas para resolver o problema, como o método de Euler, o método de Runge-Kutta e o método de *Leapfrog*, que podem fornecer soluções aproximadas com precisão suficiente para muitas aplicações práticas (HUT; MAKINO, 2007). Esses métodos discretizam o tempo em pequenos intervalos e calculam iterativamente as posições e velocidades dos corpos a cada intervalo. Alguns exemplos são:

a) Método de Euler: um método simples que assume que a velocidade e a aceleração são constantes durante cada intervalo de tempo. É computacionalmente barato, mas pode acumular erros em simulações longas (PRIGARIN; KARAVAEV; PROTASOV, 2019);
b) Método de Runge-Kutta: uma família de métodos mais precisos que usam informações sobre a derivada (aceleração) em múltiplos pontos dentro do intervalo de tempo para calcular a posição e velocidade no próximo intervalo (HUT; MAKINO, 2007);
c) Método de *Leapfrog*: um método que calcula a posição e a velocidade em intervalos de tempo intercalados, o que o torna mais estável e preciso que o método de Euler, com um custo computacional similar (QUINN, 1997).

Em geral, a simulação de N corpos envolve as seguintes etapas:

a) Definir as posições e velocidades iniciais dos objetos celestes.
b) Calcular as forças gravitacionais entre todos os pares de objetos, usando a Lei da Gravitação Universal de Newton.
c) Atualizar as velocidades e posições dos objetos com base nas forças calculadas, usando uma das abordagens numéricas mencionadas acima.
d) Repetir os passos 2 e 3 para cada intervalo de tempo desejado, até obter a evolução desejada do sistema.

Para realizar os cálculos de força gravitacional utiliza-se a fórmula da Lei da Gravitação universal de Newton para cada par de corpos no sistema (NYLONS; HARRIS; PRINS, 2007), conforme Equação 1.

$$F = G\,\frac{m_1 m_2}{r^2},\ G = 1 \tag{1}$$

Com essa força, aplica-se a Segunda Lei de Newton para determinar a aceleração de cada corpo (ROY, 1990), de acordo com a Equação 2.

$$a = \frac{F}{m} \tag{2}$$

Para finalizar, utilizam-se as fórmulas abaixo para calcular a velocidade (v), na Equação 3 e a posição (s) de cada corpo na Equação 4 (JAIN; NKOMA, 2019).

$$v = v_0 + at \tag{3}$$

$$s = s_0 + vt \tag{4}$$

Como a simulação de N corpos envolve o cálculo das interações gravitacionais entre cada par de corpos, o número de operações necessárias aumenta quadraticamente com o número de corpos (N). Esse problema de complexidade

computacional torna-se crítico em aplicações reais, que frequentemente lidam com milhões, bilhões ou até trilhões de corpos. Para contornar essa dificuldade, heurísticas e aproximações são estratégias frequentemente empregadas (WANG *et al.*, 2020).

Duas heurísticas amplamente utilizadas são o método de Barnes-Hut e o método de Partículas na Malha (PM). O método de Barnes-Hut organiza os corpos em uma estrutura de árvore octal, permitindo aproximar as forças gravitacionais exercidas por grupos distantes de corpos como se fossem um único corpo. Essa abordagem reduz a complexidade computacional para O(n log n), tornando o cálculo da força gravitacional mais eficiente (BURTSCHER; PINGALI, 2011).

Já o método PM discretiza o espaço em uma grade tridimensional e calcula a influência gravitacional em cada célula da grade utilizando a Transformada Rápida de Fourier (FFT). Essa técnica é particularmente eficiente para sistemas com grande número de corpos distribuídos uniformemente, mas pode apresentar perda de precisão em regiões com alta densidade de partículas. A escolha entre diferentes heurísticas e aproximações depende das características específicas do problema a ser simulado, como a distribuição espacial dos corpos, a precisão desejada e os recursos computacionais disponíveis (BODE; OSTRIKER; XU, 2000).

Após a aplicação dessas otimizações, a etapa de visualização e análise de dados entra em cena, permitindo extrair significado e conhecimento a partir dos resultados da simulação. Visualizações como gráficos de dispersão, animações e histogramas transcendem a representação numérica pura, possibilitando observar o movimento dos corpos, identificar padrões emergentes e obter perspectivas visuais sobre o comportamento do sistema (MCLAUGHLIN, 2006). A análise estatística, por sua vez, fornece ferramentas quantitativas para caracterizar propriedades importantes do sistema, como energia total, momento angular e distribuição espacial. A quantificação precisa dessas propriedades não só permite uma compreensão mais profunda da dinâmica do sistema, mas também serve como base sólida para validar e refinar modelos teóricos, aproximando a simulação de um reflexo fidedigno da realidade (GIERSZ; HEGGIE, 1996). A combinação sinérgica de visualização e análise estatística é, portanto, essencial para transformar os dados brutos da simulação em conhecimento acionável, impulsionando o avanço da ciência em áreas como astrofísica, cosmologia e dinâmica molecular (FUJIWARA, 2009).

2.2 REDES NEURAIS

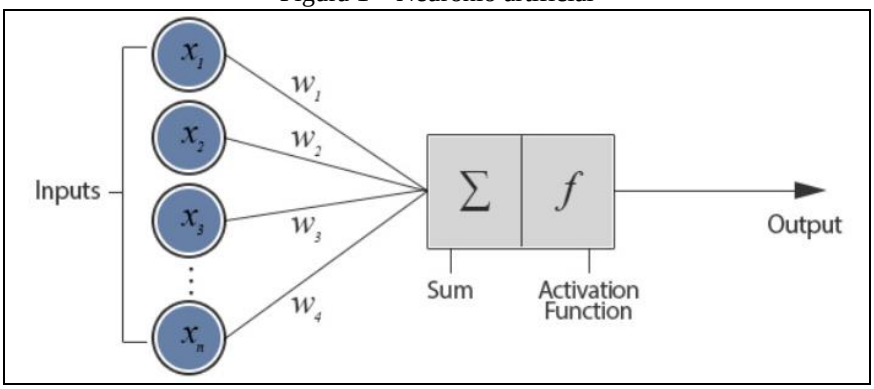
Redes Neurais Artificiais (RNAs) são modelos computacionais inspirados no cérebro humano, compostos por unidades interconectadas chamadas neurônios artificiais. Esses neurônios trabalham em conjunto para processar informações e aprender padrões a partir de dados de entrada e saída, simulando um processo de aprendizado inteligente (WU *et al.*, 2007).

O funcionamento de uma RNA, segundo Klein e Martins (2006), geralmente se divide em três etapas principais:

e) treinamento: a rede "aprende" a partir de um conjunto de dados de entrada e saída, ajustando seus parâmetros internos para minimizar erros na predição das saídas desejadas;
f) validação: com os parâmetros ajustados, a RNA é testada com um novo conjunto de dados para avaliar sua capacidade de generalização e evitar o problema de *overfitting*, isso é, decorar os dados de treinamento sem aprender os padrões;
g) utilização: uma vez que a RNA atinge um nível de precisão satisfatório na etapa de validação, ela pode ser utilizada para realizar predições e solucionar problemas similares àqueles nos quais foi treinada;

A Figura 1 ilustra a estrutura básica de uma RNA, demonstrando a organização e interconexão dos neurônios artificiais.

Figura 1 – Neurônio artificial



Fonte: Klein e Martins (2006).

Segundo Maciel (2005), uma das características das RNAs é a sua capacidade de aprender padrões complexos a partir de dados de entrada e, mais importante ainda, generalizar esse conhecimento. Isso significa que a rede consegue

interpretar e classificar novos padrões similares aos que foram apresentados durante o treinamento, mesmo que esses novos padrões não sejam idênticos aos originais. Enquanto uma rede neural do tipo *Perceptron* se limita à separação de dados linearmente separáveis, as redes *Multilayer Perceptron* (MLPs), com suas múltiplas camadas de neurônios, são capazes de aprender e representar relações não lineares entre os dados. Essa capacidade é crucial para solucionar problemas do mundo real, em que as relações entre as variáveis raramente são lineares. Contudo, o sucesso das MLPs se deve, em grande parte, ao algoritmo de aprendizado supervisionado conhecido como *backpropagation* (KLEIN; MARTINS, 2006).

O algoritmo *backpropagation* permite que a rede ajuste seus pesos e conexões de forma iterativa, buscando minimizar o erro entre suas predições e os valores reais. Esse processo ocorre em duas etapas principais. Primeiramente, na etapa de *Forward Propagation*, um dado de entrada é apresentado à rede e a informação é propagada através das camadas, calculando o valor de saída. Em seguida, na etapa de *Backward Propagation*, a saída calculada é comparada com a saída desejada (real) e a diferença entre elas, o erro, é propagada de volta através das camadas da rede. Durante essa retropropagação do erro, os pesos das conexões entre os neurônios são ajustados, buscando minimizar o erro global da rede. Esse processo de ajuste se inicia na camada de saída e se propaga até a primeira camada escondida.

O algoritmo *backpropagation*, combinado com a arquitetura MLP, tem se mostrado extremamente eficaz na resolução de problemas complexos em diversas áreas. Haykin (2001, p. 183) destaca o sucesso do modelo MLP em aplicações desafiadoras, comprovando a força do *backpropagation* como algoritmo de aprendizado supervisionado.

2.3 REDES NEURAIS CONVOLUCIONAIS

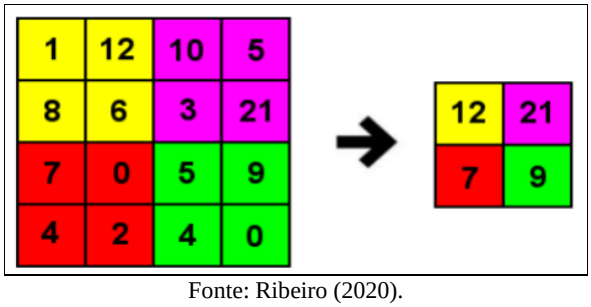
As Redes Neurais Convolucionais (CNNs) revolucionaram o campo do processamento e análise de imagens digitais. Inspiradas no sistema visual biológico, elas utilizam o conceito de campos receptivos, em que cada neurônio processa informações de uma pequena região da imagem, similar à forma como as células visuais respondem a porções específicas do campo visual. Essa característica permite que a rede explore as correlações espaciais locais presentes em imagens naturais, atuando como filtros que extraem características relevantes (Li *et al.*, 2021).

Segundo Rosa (2018), uma CNN é composta por múltiplas camadas, cada uma com uma função específica. As camadas de convolução, que dão nome à rede, são responsáveis por aplicar filtros em diferentes regiões da imagem. Cada neurônio na camada de convolução se conecta a um grupo de pixels da camada anterior, e a força dessa conexão é definida por um peso. A combinação das entradas ponderadas por seus respectivos pesos produz um valor que é passado para a próxima camada.

Após a convolução, é comum aplicar a função de ativação ReLU (*Rectified Linear Units*), que de acordo com Ribeiro (2020), aumenta a eficiência computacional da rede ao zerar os valores negativos. Em seguida, as camadas de *pooling* entram em ação, reduzindo a dimensionalidade das representações e tornando a rede mais robusta a pequenas variações na posição e aparência dos objetos na imagem.

Ribeiro (2020) destaca a técnica de *max-pooling*, ilustrada na Figura 2, que seleciona o valor máximo dentro de uma região, descartando os demais. Esse processo simplifica a informação espacial, reduzindo a quantidade de dados a serem processados e tornando a rede mais eficiente, sem perder informações cruciais sobre a presença de características importantes.

Figura 2 – Aplicação de *max-pooling* 2x2 em uma imagem 4x4



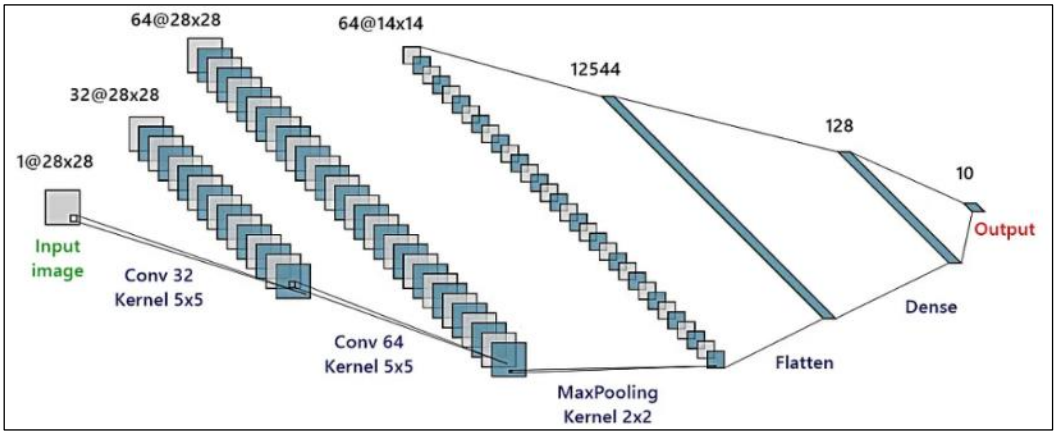
Fonte: Ribeiro (2020).

As camadas convolucionais e de *pooling* em uma CNN atuam como extratores de características, aprendendo a identificar padrões e detalhes relevantes nas imagens de entrada. À medida que a informação atravessa essas camadas, as representações se tornam mais abstratas, codificando informações sobre formas, texturas e outros elementos visuais importantes.

Segundo Ribeiro (2020), essas representações abstratas são então passadas para as camadas totalmente conectadas, responsáveis pela classificação final da imagem. Nas camadas totalmente conectadas, cada neurônio se conecta a todos os neurônios da camada anterior e a todos da camada seguinte, permitindo que a rede combine as características aprendidas pelas camadas convolucionais e de *pooling* para tomar uma decisão final sobre a classe da imagem.

A Figura 3, conforme apresentada por Soares e Carmo (2020), ilustra a variedade de filtros convolucionais utilizados para extrair diferentes características de uma imagem. Cada filtro é projetado para realçar certos padrões visuais, como bordas, cantos ou texturas. Ao aplicar esses filtros em diversas camadas e combinar suas saídas, a CNN aprende a identificar as características mais relevantes para a tarefa de classificação em questão.

Figura 3 – Arquitetura de uma Rede Neural Convolucional



Fonte: Soares e Carmo (2020).

Embora as CNNs existam há décadas, seu verdadeiro potencial só recentemente foi liberado graças aos avanços na capacidade computacional. O poder de processamento disponível atualmente permitiu o desenvolvimento e treinamento de CNNs profundas, com múltiplas camadas e milhões de parâmetros, impulsionando seu desempenho em tarefas de visão computacional.

Soares e Carmo (2020) destacam a eficácia das CNNs no processamento e análise de imagens digitais, atribuindo esse sucesso à capacidade inerente da arquitetura em capturar e analisar informações espaciais. Essa característica distintiva permite que as CNNs identifiquem padrões complexos e relações espaciais entre pixels, superando as abordagens tradicionais de visão computacional.

A validação adequada de modelos de CNNs é crucial para garantir sua generalização e evitar o *overfitting*. Uma técnica comum é a validação cruzada, em que o conjunto de dados é dividido em subconjuntos para treinamento e validação. Além disso, técnicas de regularização, como *dropout* e *weight decay*, são frequentemente empregadas para evitar que o modelo memorize os dados de treinamento ao invés de generalizar para novos exemplos.

A escolha da arquitetura, dos hiperparâmetros e das técnicas de validação depende da tarefa específica, do conjunto de dados e dos recursos computacionais disponíveis. A experimentação e a comparação sistemática de diferentes abordagens são essenciais para determinar a configuração ideal para cada problema.

2.4 REDES NEURAIS RECORRENTES

As Redes Neurais Artificiais Recorrentes (RNNs), capazes de prever dados futuros com base em exemplos passados, distinguem-se por sua capacidade única de memorização, crucial para o processamento de sequências como textos e linguagem falada (DENG; YU, 2014). Essa memória permite que a rede leve em consideração não apenas a entrada atual, mas também o histórico de informações processadas, tornando-as ideais para lidar com dados sequenciais.

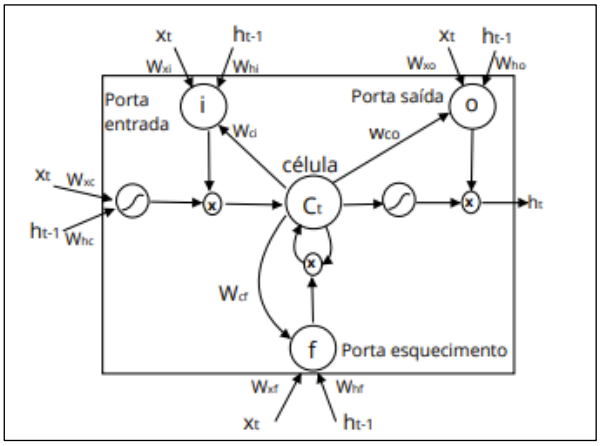
D'arbo Junior (1998) classifica as RNNs em dois tipos principais: completamente recorrentes e parcialmente recorrentes. As redes completamente recorrentes apresentam interconexões entre todas as unidades de processamento, criando um sistema altamente interdependente no qual cada unidade influencia e é influenciada por todas as outras. Em contraste, as redes parcialmente recorrentes possuem conexões mais seletivas, geralmente em uma estrutura de alimentação direta, o que simplifica o treinamento e a análise da rede.

Gomes (2005) compara a estrutura da RNN à de uma rede *feedforward*, inspirada na organização dos neurônios biológicos. Assim como no cérebro, a informação flui em uma única direção, da entrada para a saída, através de conexões ponderadas chamadas sinapses. O aprendizado da rede ocorre por meio do ajuste fino desses pesos sinápticos durante o treinamento, otimizando a capacidade da rede de prever a saída desejada para uma determinada sequência de entrada.

Para que a capacidade de modelar dependências temporais em dados sequenciais fosse possível, foram desenvolvidas algumas soluções, como o modelo de Elman (SANTANA, 2017) e o algoritmo de retropropagação através do tempo, também chamado de BPTT (D'ARBO JUNIOR, 1998). O primeiro tem limitações relacionadas ao "esquecimento" de informações importantes do passado. Já o BPTT, apesar de contornar essa questão, possui a necessidade de grande capacidade de armazenamento e tempo de processamento, especialmente para sequências longas.

Visando solucionar os problemas de memória de curto prazo do modelo de Elman, a arquitetura *Long Short-Term Memory* (LSTM) foi desenvolvida (HORCHEITER; SCHMIDHUBER, 1997). A LSTM incorpora o conceito de "células de memória", capazes de armazenar informações por períodos mais longos, e "portas" que regulam o fluxo de informações para dentro e para fora dessas células (SALAZAR, 2015). Essa estrutura, ilustrada na Figura 4, permite que a rede "aprenda" quais informações são relevantes a longo prazo e as preserve, enquanto descarta informações irrelevantes. As portas de esquecimento, entrada e saída, controladas por matrizes de pesos, garantem que a informação seja processada e armazenada de forma seletiva e eficiente (COSTA *et al.*, 2018).

Figura 4 – Bloco de memória da LSTM



Fonte: Santana (2017).

A arquitetura da LSTM, com seu fluxo de erro constante nas células de memória, oferece uma solução elegante para o problema do desaparecimento do gradiente, permitindo que a rede aprenda dependências de longo prazo de forma mais eficaz (CORRÊA, 2008). Diferentemente das RNNs tradicionais, em que o erro tende a diminuir à medida que se propaga para trás na rede, a LSTM mantém o erro constante nas células de memória, garantindo que informações relevantes do passado sejam preservadas e utilizadas no processo de aprendizado.

O treinamento da LSTM, embora similar ao das RNNs tradicionais com o uso da BPTT, apresenta uma etapa adicional crucial: a atualização dos pesos das portas de entrada, saída e esquecimento (SALAZAR, 2015). Essa atualização permite que a rede aprenda a controlar o fluxo de informações nas células de memória, decidindo quais informações devem ser armazenadas, descartadas ou utilizadas na saída.

Tanto as LSTMs quanto as RNNs tradicionais podem ser treinadas utilizando métodos de aprendizado supervisionado e não supervisionado (DENG; YU, 2014). No aprendizado supervisionado, a rede recebe pares de entrada e saída desejada, aprendendo a mapear as relações entre eles. Esse método é eficaz quando se possui um conjunto de dados rotulado, no qual cada entrada possui uma classificação predefinida. Por outro lado, o aprendizado não supervisionado explora a estrutura inerente dos dados sem a necessidade de rótulos, buscando por padrões e relações ocultas.

Em aplicações de reconhecimento de fala, por exemplo, o aprendizado não supervisionado com RNNs tem se mostrado promissor, permitindo que a rede aprenda a reconhecer padrões na fala sem a necessidade de transcrições manuais (DENG; YU, 2014). No entanto, para um desempenho ideal nesse tipo de tarefa, geralmente é necessário aumentar a complexidade da arquitetura da rede, adicionando novas camadas ocultas para capturar as nuances da linguagem.

2.5 TRABALHOS CORRELATOS

Nesta seção serão apresentados os trabalhos que correlacionam aos objetivos deste trabalho. Para realizar a busca, foi utilizada a plataforma Google Scholar. Os termos de pesquisa utilizados foram: *n-body simulation*, *three body problem*, *machine learning*, *optimization* e *graph network*. A busca combinada dessas palavras-chave retornou cerca de 160.000 registros, sendo limitado apenas a artigos criados após 2019. A escolha deles foi feita com base em sua relevância, uma medida do site que leva em conta número de citações, autores e locais de publicação.

Por fim, foram selecionados 3 artigos. O Quadro 1 descreve o uso de redes neurais profundas para o problema de N corpos, explorando seu comportamento em um ambiente caótico (BREEN *et al.*, 2020). O Quadro 2 aborda o tema por meio de redes neurais orientadas a grafos (GONZALEZ *et al.*, 2020). No Quadro 3 será apresentado o desenvolvimento de uma rede neural profunda aplicada a conceitos de mapa de densidade e vetores de deslocamento (OLIVEIRA *et al.*, 2020).

Quadro 1 – Newton versus the machine: solving the chaotic three-body problem using deep neural networks

Referência	Breen *et al.* (2020)
Objetivos	Desenvolver uma abordagem para prever a trajetória de partículas em encontros próximos de três corpos de mesma massa utilizando uma RNA.
Principais funcionalidades	Prever a trajetória de partículas durante encontros próximos. Minimizar erros em simulações de três corpos.
Ferramentas de desenvolvimento	RNA. Uso do integrador numérico Brutus para geração dos dados de treino.
Resultados e conclusões	A taxa de energia relativa com MAE inferior a 0.1%. Erros médios na taxa de 10^{-5} em encontros próximos ao utilizar uma camada de projeção. A RNA reproduziu comportamento caótico esperado.

Fonte: elaborado pelo autor.

Quadro 2 – Learning to Simulate Complex Physics with Graph Networks

Referência	Gonzalez *et al.* (2020)
Objetivos	Desenvolver um framework de aprendizado de máquina orientado a grafos para simulação de partículas em sistemas físicos
Principais funcionalidades	*Encoder*: Cria um grafo no qual cada nó representa uma partícula e conexões são recalculadas a cada *time-step* usando o algoritmo do vizinho mais próximo. *Processor*: Processa interações entre nós com M camadas de passagem de mensagens e conexões residuais. *Decoder*: Uma *perceptron* multicamadas que extrai a aceleração das partículas e atualiza o sistema utilizando integração de Euler.
Ferramentas de desenvolvimento	Rede neural orientada a grafos (*Graph Neural Network* - GNN). Ferramentas de simulação para geração dos dados de treino: BOXBATH, SPlisHSPlasH, Taichi-MPM.
Resultados e conclusões	Simulações com EQM na ordem de 10^{-3}, variando conforme os materiais e suas interações. O modelo conseguiu entender bem a dinâmica entre diferentes materiais e realizar previsões plausíveis a longo prazo. Comparado com outros modelos, a solução proposta foi mais simples e exata. Possibilidade de otimizações futuras com computação paralela.

Fonte: elaborado pelo autor.

Quadro 3 – Fast and Accurate Non-Linear Predictions of Universes with Deep Learning

Referência	Oliveira *et al.* (2020)
Objetivos	Abordar o problema das aproximações em simulações de N corpos em escalas menores, em que a flutuação da densidade das partículas se torna significativa.
Principais funcionalidades	Rede Neural Convolucional (*Convolutional Neural Network* - CNN). Trabalhar com três resoluções diferentes do problema, com camadas de *downsampling* e *upsampling*. Utilizar conexões residuais em cada camada de convolução.
Ferramentas de desenvolvimento	CNN com arquitetura V-NET. Uso dos dados das simulações do Projeto Quijote.
Resultados e conclusões	Modelo treinado com 210 simulações, alcançando mais de 99% de acurácia. Tempo de treinamento duas vezes mais rápido que métodos tradicionais. Melhor desempenho em prever flutuações de densidade em comparação aos métodos tradicionais. O estudo pode permitir simulações complexas do universo com alta precisão em computadores pessoais.

Fonte: elaborado pelo autor.

3 DESCRIÇÃO DO ARTEFATO COMPUTACIONAL

Para melhor entendimento do desenvolvimento do artefato computacional desenvolvido, subdividiu-se este capítulo em três seções. A seção 3.1 abordará sobre o desenvolvimento do simulador de N corpos responsável por gerar a base de dados. A seção 3.2 apresentará a ferramenta de visualização das simulações. E, por fim, a seção 3.3 irá abordar o desenvolvimento do modelo de inteligência artificial de N corpos.

3.1 DESCRIÇÃO DO SIMULADOR

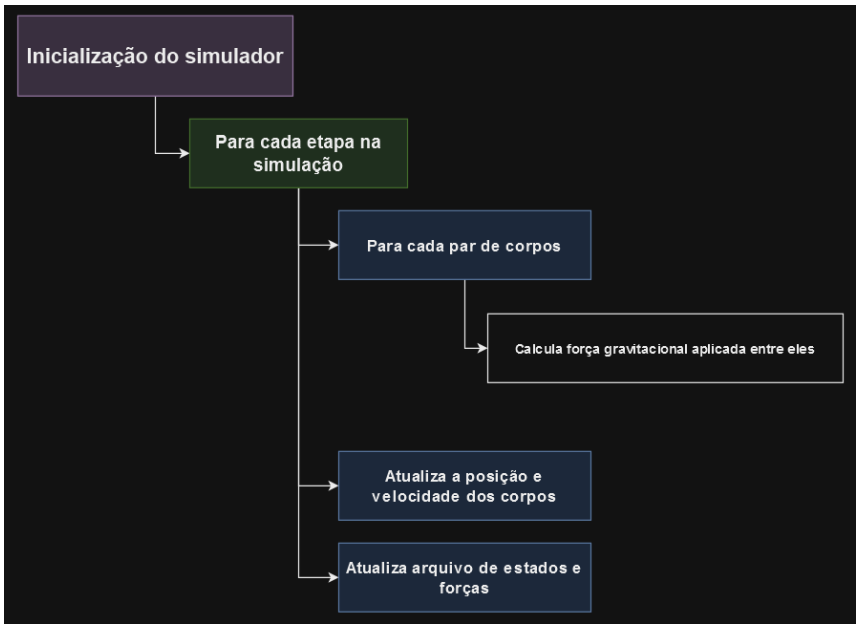
Nessa seção serão abordados os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF), assim como os principais componentes do simulador de N corpos.

O simulador deverá:

a) processar uma simulação utilizando forças gravitacionais dado as configurações iniciais para os corpos, incluindo posições, massas, velocidades iniciais e duração do *time-step* (RF);
b) gerar como resultado dois arquivos: um contendo o estado de cada uma das partículas em cada etapa da simulação, outro com a força aplicada em cada partícula em cada etapa da simulação (RF);
c) ser capaz de reproduzir soluções periódicas já encontradas para o problema de três corpos (RNF);
d) ser implementado utilizando a linguagem de programação Rust (RNF);

A Figura 5 ilustra o fluxo de funcionamento do simulador de interações gravitacionais entre corpos. O processo se inicia com a definição dos parâmetros da simulação, como número de corpos, suas propriedades iniciais e detalhes do processo iterativo. Em seguida, para cada passo de tempo, o simulador calcula as forças gravitacionais mútuas entre cada par de corpos, atualiza suas posições e velocidades e registra esses dados em arquivos para análise posterior.

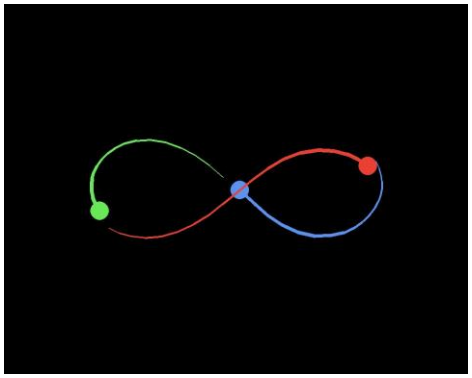
Figura 5 – Fluxo de funcionamento do simulador



Fonte: elaborado pelo autor

Ressalta-se que para validar o simulador, bem como a precisão dos cálculos realizados em cada etapa, recorreu-se ao estudo de Chenciner & Montgomery (2000). Os autores definiram um estado inicial específico para o problema de três corpos que resulta em uma solução periódica, gerando uma configuração visual similar ao símbolo do infinito, como apresentado na Figura 6. A reprodução dessa solução específica pelo simulador demonstra sua capacidade de modelar corretamente as interações gravitacionais e gerar resultados precisos para diferentes configurações.

Figura 6 – Modelo de simulação de Chenciner & Montgomery (2000)



Fonte: elaborado pelo autor

Inicialmente, o simulador recebe como entrada o estado inicial da simulação, duração do *time-step*, a constante gravitacional, o parâmetro de amortecimento e a quantidade de etapas a serem executadas. O estado inicial é composto por uma lista de corpos, cada um com propriedades de posição e velocidade, representadas por vetores 3D, além da massa. A duração do *time-step* é um *float* que indica o tempo (em segundos) que cada etapa da simulação representa. A constante gravitacional faz parte da fórmula de força gravitacional aplicada à cada corpo. O parâmetro de amortecimento serve para

atenuar encontros próximos entre corpos, evitando que as forças cresçam demais. O Quadro 4 exemplifica a passagem de parâmetros para o simulador.

Quadro 4 – Parâmetros de inicialização do simulador

```
simulador = Simulador(
    estado_inicial = [
        corpo(
            posicao = (-0.97, 0.24, 0),
            velocidade = (0.46, 0.43, 0),
            massa = 1,
        ),
        corpo(
            posicao = (0, 0, 0),
            velocidade = (-0.93, -0.86, 0),
            massa = 1,
        ),
        corpo(
            posicao = (0.97, -0.24, 0),
            velocidade = (0.46, 0.43, 0),
            massa = 1,
        ),
    ],
    duracao_time_step = 0.1,
    total_etapas = 100,
    constante_gravitacional = 1,
    parametro_amortecimento = 0.00001,
)
```

Fonte: elaborado pelo autor

No Quadro 5 é apresentado um pseudocódigo com a lógica principal do simulador, com complexidade quadrática. Como a força entre dois corpos é equivalente, mas em direções opostas, pode-se otimizar a iteração aplicando a força a ambos os corpos simultaneamente. Embora a complexidade algorítmica se mantenha, isso torna o processo ligeiramente mais eficiente. Todavia, o código no Quadro 5 simula a interação gravitacional entre um conjunto de corpos ao longo do tempo. A simulação é dividida em uma série de etapas. Em cada etapa, o código calcula a força gravitacional entre cada par único de corpos e atualiza suas velocidades e posições com base nessa força.

Quadro 5 – Pseudocódigo do funcionamento interno do simulador

```
para _ entre 0 e total_etapas:
    para i entre 0 e lista_corpos.tamanho:
        corpo1 = lista_corpos[i]
        para j entre i + 1 e lista_corpos.tamanho:
            corpo2 = lista_corpos[j]
            gravidade = calcular_gravidade(corpo1, corpo2)
            corpo1.forca = corpo1.forca + gravidade
            corpo2.forca = corpo2.forca - gravidade
        aceleração = corpo1.forca / corpo1.massa
        corpo1.velocidade = corpo1.velocidade + aceleração * time_step
        corpo1.distância = corpo1.distância + corpo1.velocidade * time_step
        corpo1.força = [0, 0, 0]
    salvar_arquivo_forcas()
    salvar_arquivo_estados()
```

Fonte: elaborado pelo autor

A partir do Quadro 5, observa-se que o código define uma função chamada `calcular_gravidade`, que utiliza a lei da gravitação de Newton para determinar a força entre dois corpos quaisquer. Segundo a fórmula da Gravitação Universal citada na seção 2.1, à medida que a distância se aproxima de zero a força tende ao infinito. Para contornar esse problema, se utiliza o parâmetro de amortecimento, um valor fixo adicionado à distância entre os objetos que impede que o denominador da função seja zero. Idealmente, é preferível usar um *time-step* dinâmico para capturar de forma mais acurada as forças aplicadas em momentos críticos. Entretanto, devido à essência discreta do algoritmo, ambas as soluções acabam sendo uma aproximação do problema. Com o uso do parâmetro de amortecimento, o cálculo de gravidade segue a estrutura apresentada no Quadro 6.

Quadro 6 – Pseudocódigo para cálculo da força gravitacional

```
função calcular_gravidade(corpo1, corpo2):
    distancia = corpo2.posicao – corpo1.posicao
    distancia_2 = distancia * distancia + amortecimento * amortecimento
    distancia_3 = raiz(distancia_2) * distancia_2

    forca = constante_gravitacional * (corpo1.massa * corpo2.massa) / distancia_3
    retorna forca * distancia
```

Fonte: elaborado pelo autor

Em seguida, o código do Quadro 5 entra em um loop principal que itera um número predeterminado de vezes, representando as etapas da simulação. Dentro deste *loop*, o código percorre todos os corpos na simulação. Para cada corpo, ele calcula a força gravitacional exercida sobre ele por todos os outros corpos e a adiciona à sua força total. Uma vez que a força total em um corpo é conhecida, sua aceleração é calculada usando a segunda lei de Newton, conforme Equação 5. Esta aceleração é então usada para atualizar a velocidade e a posição do corpo.

$$F = ma \tag{5}$$

Finalmente, após cada etapa da simulação, o código salva as informações sobre as forças e os estados dos corpos em dois arquivos: um contendo os estados e outro com as forças aplicadas a cada corpo. Em ambos, cada linha representa o estado "n" da simulação. No Quadro 7, é apresentado o formato do arquivo de estados. Ele é composto por uma lista de objetos, em que cada item representa um dos corpos com suas respectivas propriedades: `posição`, `velocidade` e `massa`.

Quadro 7 – Formato do arquivo de estados do simulador

```
[{"pos": [0, 0, 0], "vel": [0, 0, 0], "mass": 1}, {…}, …]
```

Fonte: elaborado pelo autor

No Quadro 8 está representado o formato do arquivo de forças. Ele consiste em uma lista de vetores, em que cada item representa um dos corpos com a força aplicada a ele nos eixos `x`, `y` e `z`.

Quadro 8 – Formato do arquivo de forças do simulador

```
[[1, 1, 1], [2, 2, 2], [3, 3, 3], …]
```

Fonte: elaborado pelo autor

Embora o simulador realize cálculos para qualquer número de corpos, ele foi projetado para rodar para uma baixa quantidade. Por essa razão, não utiliza nenhuma heurística que reduza sua complexidade algorítmica nem paralelismos na GPU. Portanto, não é recomendado para o processamento de um alto volume de corpos.

3.2 DESCRIÇÃO DO VISUALIZADOR

Tendo apresentado o simulador de N corpos e seus principais componentes na seção anterior. Esta seção detalha os requisitos funcionais e não funcionais que guiaram o desenvolvimento do visualizador, além de descrever os principais componentes que permitem a visualização dinâmica e interativa da simulação de N corpos.

O visualizador deverá:

a) gerar uma visualização em que cada corpo seja representado por um círculo, exibindo um rastro que ilustre seu movimento (RF);
b) entender o formato do arquivo de estados gerado pelo simulador e processar uma linha por *frame* (RF);
c) ser implementado utilizando a linguagem de programação Python e a biblioteca Pygame (RNF);

O visualizador recebe como entrada o arquivo de estados gerado pelo simulador descrito no Quadro 7. A cada *frame*, uma linha do arquivo é processada para renderizar o estado atual da simulação. Com base na posição de cada corpo, um círculo é desenhado na posição `x` e `y`. Essas posições são salvas em um *array* em tempo de execução para apresentar o rastro da movimentação de cada corpo. Esse processo está ilustrado no Quadro 9.

Quadro 9 – Pseudocódigo do funcionamento interno da ferramenta de visualização

```
rastros = []
frame = 0
arquivo = ler(simulação)
enquanto não_esta_fim_arquivo():
    corpos = carregar_proxima_linha(arquivo)

    para corpo em corpos:
        renderizar_corpo(corpo, tela)

    atualizar_lista_rastros(corpos)
    renderizar_rastros(rastros, tela)
    salvar_imagem_frame(tela, frame)
    frame = frame + 1
```

Fonte: elaborado pelo autor

Existem alguns parâmetros que permitem customizar a forma como as informações serão exibidas. São eles: largura, altura, *zoom*, raio do corpo, grossura do rastro e a duração máxima do rastro. A largura e altura definem o tamanho da tela. O raio do corpo determina o tamanho dos círculos que representam os corpos. A grossura do rastro define a espessura máxima do rastro. A inicialização do visualizador é apresentada no Quadro 10.

Quadro 10 – Parâmetros de inicialização da visualização

```
visualização = Visualização(
    arquivo_estados = "simulação_1.txt",
    largura = 1000,
    altura = 800,
    zoom = 100,
    raio_corpo = 10,
    grossura_rastro = 10,
    duração_maxima_rastro = 25
)
```

Fonte: elaborado pelo autor

A duração máxima do rastro especifica quantas posições anteriores serão exibidas. As posições do rastro são armazenadas em uma fila que guarda as últimas x posições, em que x é o valor desse parâmetro. Quando o tamanho da fila é excedido, o primeiro item adicionado é removido para dar lugar a uma nova posição. Esse algoritmo está descrito no Quadro 11.

Quadro 11 – Pseudocódigo com a lógica de atualização da lista de rastros

```
rastros = []

função atualizar_lista_rastros(corpos):
    posições = []
    para corpo em corpos:
        posições.adicionar([corpo.x, corpo.y])
    rastros.enfileirar(posições)
    se rastros.tamanho > duração_maxima_rastro:
        rastros.desenfileirar()
```

Fonte: elaborado pelo autor

Para a renderização dos corpos e dos rastros, utilizam-se os algoritmos descritos no Quadro 12.

Quadro 12 – Pseudocódigo com a lógica de renderização dos corpos e rastros usando a biblioteca Pygame

```
função renderizar_corpo(corpo, tela):
    pygame.draw.circle(
        surface = tela,
        color = (255, 255, 255),
        center = (corpo.x, corpo.y),
        radius = raio_corpo
    )

função renderizar_rastros(rastros, tela):
    para i entre 1 e rastros.tamanho:
        para j entre 0 e rastros[i].tamanho:
            pygame.draw.line(
                surface = tela,
                color = (255, 255, 255),
                start_pos = (rastros[i][j][0], rastros[i][j][1]),
                end_pos = (rastros[i - 1][j][0], rastros[i - 1][j][1]),
                width = grossura_rastro * (1 - i / duração_maxima_rastro)
            )
```

Fonte: elaborado pelo autor

Conforme apresentado no Quadro 12, a função `renderizar_corpo` utiliza as coordenadas x e y dos corpos para desenhá-los na tela, bem como o parâmetro `raio_corpo` para determinar seu tamanho. A função `renderizar_rastros`, por sua vez, descreve o algoritmo responsável por desenhar os rastros dos corpos. Nesse processo, linhas são traçadas entre pontos consecutivos de um mesmo corpo, sendo sua espessura inversamente proporcional à sua posição na fila de rastros, ou seja, quanto mais antiga a posição no rastro, mais fina a linha. Como resultado, a simulação é executada em tempo real, e a cada frame, uma imagem é capturada e salva em uma pasta específica, conforme detalhado no Quadro 13. Esse processo permite a análise posterior da evolução do sistema, fornecendo um registro visual completo da simulação.

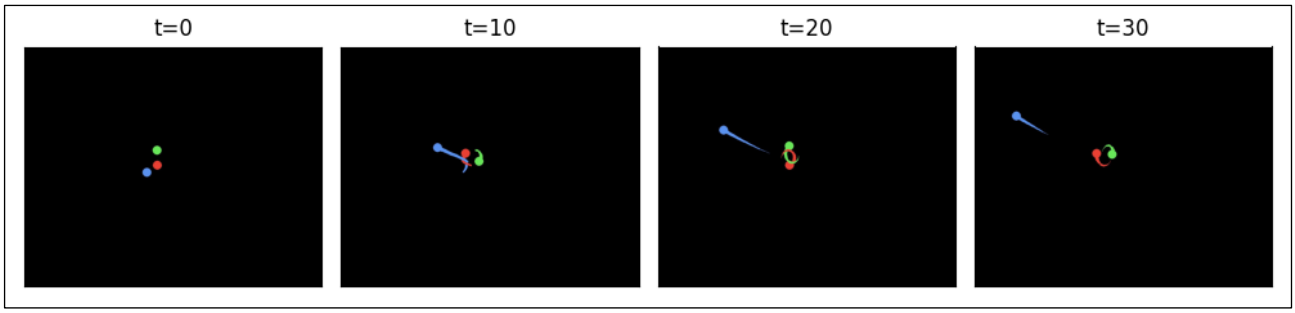
Quadro 13 – Pseudocódigo com o salvamento dos frames usando a biblioteca Pygame

```
função salvar_imagem_frame(tela, numero_frame):
    pygame.image.save(tela, "./screenshots/screenshot_" + numero_frame + ".jpg")
```

Fonte: elaborado pelo autor

A Figura 7 mostra um exemplo de visualização gerada a partir de uma única simulação em quatro momentos diferentes: o primeiro, o décimo, o vigésimo e o trigésimo *frame*. Cada círculo representa um corpo (identificado por cores distintas) e o trajeto de cada corpo é mostrado pelas curvas correspondentes da mesma cor.

Figura 7 – Exemplo de simulação apresentada pelo visualizador em 4 momentos diferentes



Fonte: elaborado pelo autor

Com base na Figura 7, observa-se que os corpos se movem sob influência da atração gravitacional mútua, exibindo variações de velocidade, evidenciadas pela extensão dos rastros entre os instantes de tempo. A dança gravitacional é marcada por momentos de aproximação e afastamento entre os corpos. A ausência de informações sobre a massa dos corpos e as escalas de tempo e espaço limita uma análise quantitativa mais aprofundada. No entanto, a sequência de instantes sugere um sistema dinâmico potencialmente caótico, em que pequenas alterações nas condições iniciais poderiam resultar em trajetórias significativamente distintas. A visualização, nesse contexto, torna-se crucial para a compreensão da dinâmica do sistema, permitindo identificar padrões de movimento, interações relevantes e formular hipóteses sobre seu comportamento futuro. Neste sentido, a ferramenta de visualização se mostrou uma grande aliada para o entendimento do problema de três corpos. Por ter sido desenvolvida especificamente para este contexto, estratégias de otimização no processo de renderização e leitura de arquivos não foram implementadas. Consequentemente, o uso da ferramenta não é recomendado para cenários com um alto número de corpos.

3.3 DESCRIÇÃO DO MODELO

Após a apresentação e discussão das características do simulador e do visualizador, esta seção detalha os modelos desenvolvidos. Primeiramente, serão abordados os requisitos funcionais e não funcionais, bem como os principais componentes do modelo de inteligência artificial.

O modelo a ser desenvolvido deverá:

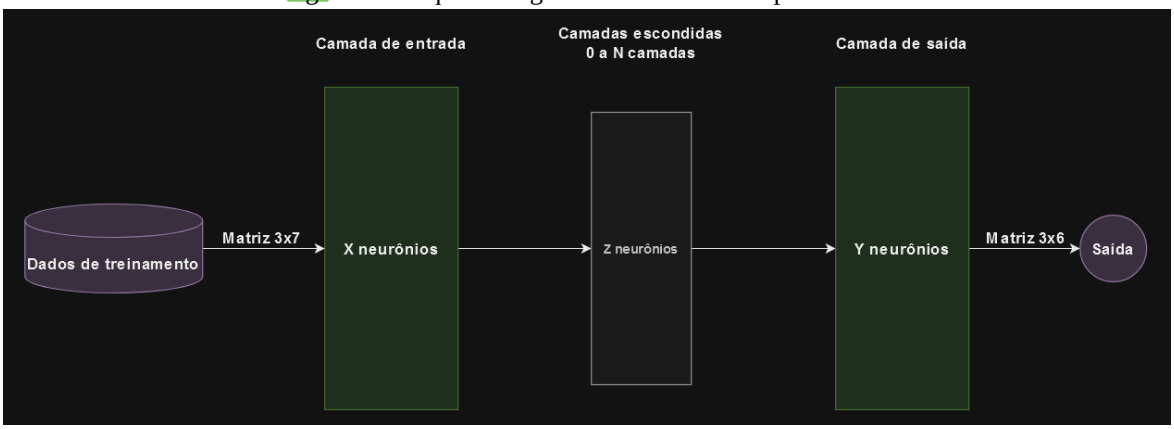
e) ser capaz de prever o movimento de três corpos dado as suas condições iniciais (RF);
f) ser capaz de entender os arquivos gerados pelo simulador (RF);
g) gerar como resultado as forças aplicadas à cada corpo (RF);
h) ser capaz de reproduzir o comportamento caótico do problema (RNF);
i) ser desenvolvido em Python (RNF);

Ressalta-se que o modelo passou por uma fase inicial de investigação e experimentação antes de ser desenvolvido em sua forma definitiva. Essa etapa exploratória foi crucial para compreender as limitações e necessidades inerentes ao problema, permitindo, posteriormente, a busca por soluções mais eficazes. Esses dois processos, investigação e desenvolvimento, são detalhados nas seções 3.3.1 e 3.3.2, respectivamente.

3.3.1 MODELO INICIAL

Na primeira abordagem do problema, considerou-se uma arquitetura com matriz de entrada 3x7, em que 3 é o número de corpos e 7 o número de propriedades de cada corpo. As propriedades são posição (`x`, `y` e `z`), velocidade (`x`, `y` e `z`) e massa. Como saída gerou-se uma matriz 3x6 com as posições e velocidades de cada corpo atualizadas. Nessa perspectiva, foram explorados diversos algoritmos e arquiteturas de aprendizado de máquina, incluindo RNAs, CNNs, RNNs e aprendizado por reforço. Cada abordagem foi avaliada com diversos hiperparâmetros, números de camadas, funções de ativação, entre outros. A Figura 8 ilustra a arquitetura geral dos modelos de aprendizado.

Figura 8 – Arquitetura geral dos modelos de aprendizado



Fonte: elaborado pelo autor

O formato de entrada está explicitado no Quadro 14, a primeira linha apresenta a estrutura da entrada e na segunda, um exemplo com o estado inicial proposto por Chenciner & Montgomery (2000).

Quadro 14 – Entrada do modelo

Estrutura de entrada	`[`
	`  [pos_x1, pos_y1, pos_z1, vel_x1, vel_y1, vel_z1, massa1],`
	`  [pos_x2, pos_y2, pos_z2, vel_x2, vel_y2, vel_z2, massa2],`
	`  [pos_x3, pos_y3, pos_z3, vel_x3, vel_y3, vel_z3, massa3],`
	`]`

Exemplo do estado inicial	`[`
	`  [-0.97000436, 0.24308753, 0, 0.4662036850, 0.4323657300, 0, 1],`
	`  [0, 0, 0, -0.93240737, -0.86473146, 0, 1],`
	`  [0.97000436, -0.24308753, 0, 0.4662036850, 0.4323657300, 0, 1],`
	`]`

Fonte: elaborado pelo autor

O formato da saída está apresentado no Quadro 15. A primeira linha descreve a estrutura da saída, enquanto a segunda linha ilustra um exemplo real retornado pelo modelo com base na entrada fornecida no Quadro 14.

Quadro 15 – Saída do modelo

Estrutura da saída	[
	  [pos_x1, pos_y1, pos_z1, vel_x1, vel_y1, vel_z1, massa1],
	  [pos_x2, pos_y2, pos_z2, vel_x2, vel_y2, vel_z2, massa2],
	  [pos_x3, pos_y3, pos_z3, vel_x3, vel_y3, vel_z3, massa3],
	]

Exemplo real retornado pelo modelo	[
	  [-1.440091, 0.513950, 0.096934, -0.345301, 0.306915, 0.246708],
	  [0.987223, 0.026510, 0.289608, 0.067298, 0.389701, 0.037642],
	  [-0.453717, 0.062839, -0.066489, 0.061361, -0.620889, 0.199026],
	]

Fonte: elaborado pelo autor

Para geração do conjunto de dados de treino, utilizou-se o simulador descrito na seção 3.1. No total, foram geradas 101 mil transições de estado. As posições e a velocidade foram geradas aleatoriamente dentro do intervalo de -1 e 1. A massa, também aleatória no intervalo de 0 até 1. Esses intervalos foram definidos no momento da geração dos dados para evitar a necessidade de normalização dos valores durante a etapa de pré-processamento. Essa etapa inclui a conversão do formato de arquivo gerado pelo simulador para uma matriz 2D, conforme detalhado no Quadro 16.

Quadro 16 – Algoritmo de conversão do arquivo de estados para matriz 2D

```
estado_inicial = carregar_arquivo_estados()
entrada_modelo = []
para corpo em estado_inicial:
    entrada_modelo.adicionar([
        corpo.pos.x, corpo.pos.y, corpo.pos.z,
        corpo.vel.x, corpo.vel.y, corpo.vel.z,
        corpo.mass
    ])
```

Fonte: elaborado pelo autor

A seguir, são descritas as versões mais promissoras de algoritmos e arquiteturas de aprendizado de máquina, incluindo RNAs, CNNs, RNNs e aprendizado por reforço. A arquitetura da RNA implementada consiste em uma camada de entrada, três camadas ocultas com 64 nós cada e uma camada de saída. A função de ativação ReLU foi aplicada às camadas ocultas. O modelo foi compilado utilizando o otimizador Adam, com taxa de aprendizado inicial de 0,001 e a função de perda EQM (Erro Quadrático Médio). Durante o treinamento, foram utilizados um tamanho de lote (*batch size*) de 1000 amostras e uma divisão de 15% dos dados para validação. Essa estrutura está apresentada na Figura 9.

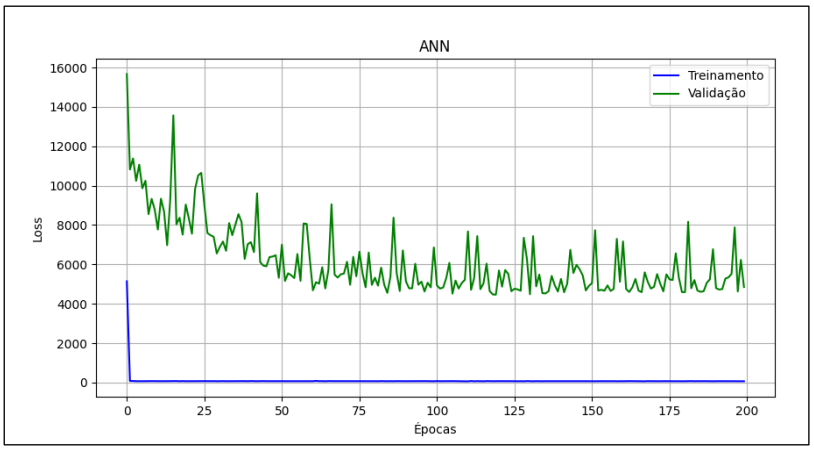
Figura 9 – Parâmetros RNA

```python
class NBodyANNModel(tf.keras.Model):
    def __init__(self, **kwargs):
        super(NBodyANNModel, self).__init__(**kwargs)
        self.dense1 = tf.keras.layers.Dense(64, activation="relu")
        self.dense2 = tf.keras.layers.Dense(64, activation="relu")
        self.dense3 = tf.keras.layers.Dense(64, activation="relu")
        self.dense4 = tf.keras.layers.Dense(6)


def train_ann_model(X, y, epochs):
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

    model = NBodyANNModel()
    model.compile(optimizer=optimizer, loss="mse")
    training = model.fit(X, y, batch_size=1_000, validation_split=0.15, epochs=epochs)
```

Fonte: elaborado pelo autor.

Ao final de 200 épocas, o modelo teve *loss* de treinamento e validação igual a 64,77 e 4851,21, respectivamente. A Figura 10 apresenta a evolução desses valores no decorrer de cada época.

Figura 10 – Evolução da *loss* de treinamento e validação - RNA



Fonte: elaborado pelo autor.

A Figura 10 demonstra que o modelo convergiu para uma resposta próximo à época 70, sem apresentar mudanças significativas após esse ponto. No entanto, a discrepância observada entre as curvas de perda (*loss*) de validação e de treinamento indica que a RNA encontrou dificuldades em generalizar os resultados para novos dados. Essa diferença acentuada sugere um caso de *overfitting*, ou seja, o modelo se ajustou muito bem aos dados de treinamento, mas não conseguiu generalizar o aprendizado para dados não vistos anteriormente.

Para a CNN, utilizou-se uma arquitetura com uma camada de entrada, duas camadas de convolução contendo 32 filtros e *kernel* de tamanho 3, três camadas ocultas contendo 64 nós e uma camada de saída. A função de ativação escolhida para as camadas de convolução e ocultas foi a ReLU. Para compilação do modelo foi utilizado o otimizador Adam com taxa de aprendizado inicial 0,001 e EQM como função de *loss*. Para o treinamento empregou-se *batch_size* 1000 e divisão de 15% dos dados para validação. Essa configuração está disposta na Figura 11.

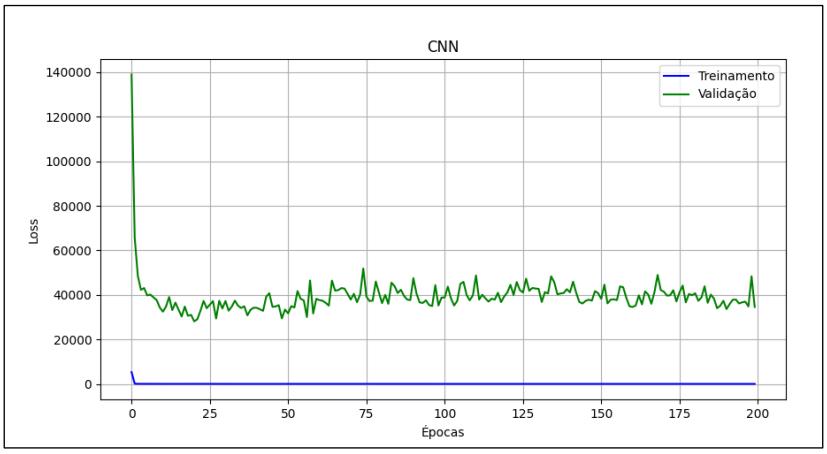
Figura 11 – Parâmetros CNN

```
class NBodyCNNModel(tf.keras.Model):
    def __init__(self):
        super(NBodyCNNModel, self).__init__()
        self.num_filters = 32
        self.kernel_size = 3
        self.num_layers = 2

        self.conv_layers = [
            tf.keras.layers.Conv1D(
                filters=self.num_filters,
                kernel_size=self.kernel_size,
                activation="relu",
                padding="same",
            )
            for _ in range(self.num_layers)
        ]
        self.dense1 = tf.keras.layers.Dense(64, activation="relu")
        self.dense2 = tf.keras.layers.Dense(64, activation="relu")
        self.dense3 = tf.keras.layers.Dense(64, activation="relu")
        self.dense4 = tf.keras.layers.Dense(6)


def train_cnn_model(X, y, epochs):
    optimizer = tf.keras.optimizers.Adam()

    model = NBodyCNNModel()
    model.compile(optimizer=optimizer, loss="mse")
    training = model.fit(X, y, batch_size=1_000, validation_split=0.15, epochs=epochs)
```

Fonte: elaborado pelo autor.

Ao final de 200 épocas, o modelo teve *loss* de treinamento e validação igual a 73,76 e 34.600,23, respectivamente. A Figura 12 apresenta a evolução desses valores no decorrer de cada época

Figura 12 – Evolução da *loss* de treinamento e validação - CNN



Fonte: elaborado pelo autor.

O resultado da Figura 12 mostra que próximo da época 75 o modelo convergiu para uma resposta. Além disso, nota-se a diferença entre *loss* de treino e validação, um sinal de que a CNN teve dificuldade de generalizar novos resultados.

Para a RNN, utilizou-se uma arquitetura com uma camada de entrada, duas camadas LSTM contendo 64 nós, três camadas ocultas contendo 64 nós e uma camada de saída. A função de ativação escolhida para as camadas LSTM e ocultas foi a ReLU. Para compilação do modelo foi utilizado o otimizador Adam com taxa de aprendizado inicial 0,001 e EQM como função de *loss*. Para o treinamento foi aplicado um *batch_size* igual a 1000 e divisão de 15% dos dados para validação. Esta arquitetura está ilustrada na Figura 13.

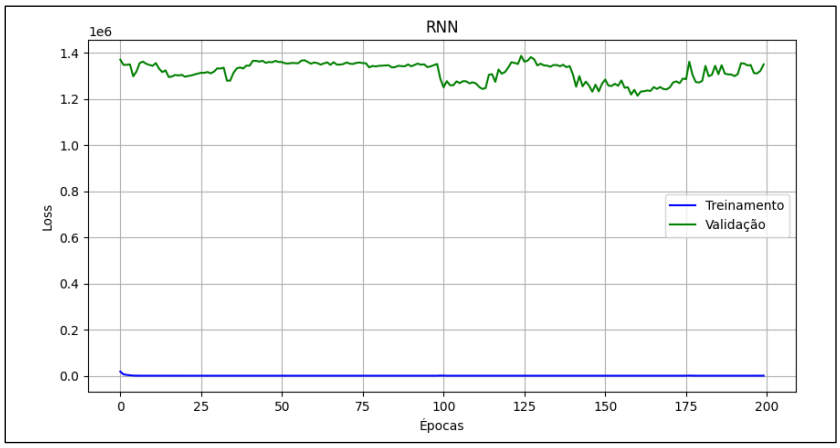
Figura 13 – Parâmetros RNN

```
class NBodyRNNModel(tf.keras.Model):
    def __init__(self, **kwargs):
        super(NBodyRNNModel, self).__init__(**kwargs)
        self.hidden_units = 64
        self.num_layers = 2

        self.rnn_layers = [
            tf.keras.layers.LSTM(units=self.hidden_units, return_sequences=True)
            for _ in range(self.num_layers)
        ]
        self.dense1 = tf.keras.layers.Dense(64, activation="relu")
        self.dense2 = tf.keras.layers.Dense(64, activation="relu")
        self.dense3 = tf.keras.layers.Dense(64, activation="relu")
        self.dense4 = tf.keras.layers.Dense(6)


def train_rnn_model(X, y, epochs):
    optimizer = tf.keras.optimizers.Adam()

    model = NBodyRNNModel()
    model.compile(optimizer=optimizer, loss="mse")
    training = model.fit(X, y, batch_size=1_000, validation_split=0.15, epochs=epochs)
```

Fonte: elaborado pelo autor.

Ao final de 200 épocas, o modelo teve *loss* de treinamento e validação igual a 267,82 e 1.349.339,88, respectivamente. A Figura 14 apresenta a evolução desses valores no decorrer de cada época.

Figura 14 – Evolução da *loss* de treinamento e validação - RNN



Fonte: elaborado pelo autor.

O resultado da Figura 14 mostra que logo nas primeiras épocas o modelo conseguiu convergir para uma resposta, sendo de longe o pior dentre as três alternativas. Assim como os outros, por conta da diferença das *losses*, percebe-se que a RNN teve dificuldade de generalizar novos resultados.

Quanto ao aprendizado por reforço, desenvolveu-se uma prova de conceito, pois muitos algoritmos se baseiam em um espaço de estados/ações discreto, ao passo que o problema de N corpos trabalha com um espaço contínuo. Foi utilizado o algoritmo *Actor-Critic*, em que são utilizadas duas RNAs, uma para tomar a ação e outra para calcular a recompensa. No entanto, devido à complexidade adicional, redundância frente às outras estratégias e ao custo computacional envolvido, essa abordagem foi descartada.

A ferramenta de visualização também foi utilizada para comparar a influência do efeito caótico em cada um dos modelos descritos acima. Para tanto, inicialmente, foi necessário implementar um algoritmo que convertesse a saída do modelo para um formato compreensível pelo visualizador. Esse algoritmo está descrito no Quadro 17.

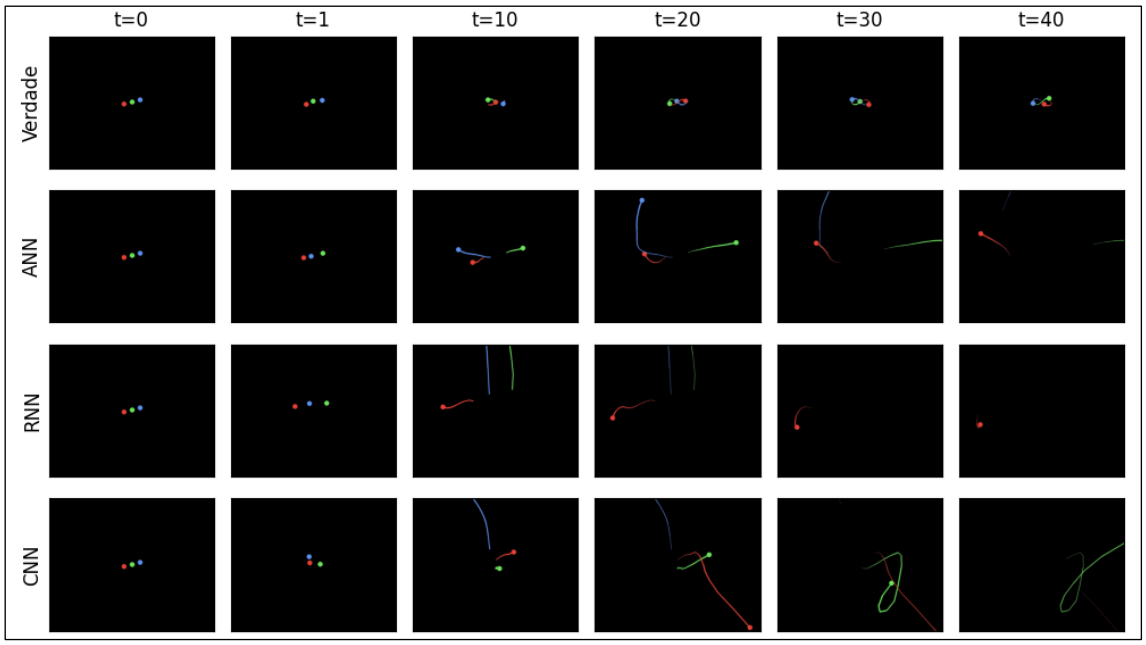
Quadro 17 – Algoritmo de conversão da saída do modelo para o arquivo de estados

```
estado_inicial = carregar_estado_inicial()
estado_previsto = []
para i, corpo em saída_modelo:
    estado_previsto.adicionar({
        "pos": [corpo[0], corpo[1], corpo[2]],
        "vel": [corpo[3], corpo[4], corpo[5]],
        "mass": estado_inicial[i].mass
    })
```

Fonte: elaborado pelo autor

Por fim, aplicou-se a simulação em formato de oito de Chenciner e Montgomery (2000) de forma recursiva em cada um deles. A Figura 15 apresenta os resultados obtidos em 6 tempos diferentes.

Figura 15 – Comparação do efeito caótico sobre a ANN, RNN e CNN



Fonte: elaborado pelo autor

A Figura 15 mostra uma comparação de diferentes modelos de aprendizado de máquina na previsão da trajetória de um sistema de três corpos ao longo do tempo. Diante das simulações da trajetória, pode se fazer as seguintes análises:

a) Verdade: a trajetória real mostra os corpos se movendo em um padrão orbital complexo devido à interação gravitacional entre eles;
b) ANN: a Rede Neural Artificial consegue acompanhar a trajetória real por um curto período, mas depois começa a divergir, mostrando imprecisão em previsões de longo prazo;
c) RNN: a Rede Neural Recorrente apresenta o pior desempenho entre os modelos, falhando em capturar o movimento orbital e divergindo da trajetória real logo no início;
d) CNN: a Rede Neural Convolucional assim como a ANN, teve dificuldade de manter a trajetória correta por um longo período;

Contudo, pode-se perceber que, para este problema específico de previsão de trajetória de múltiplos corpos, a ANN apresenta melhor desempenho que a RNN e a CNN. No entanto, nenhum dos modelos testados consegue replicar perfeitamente a trajetória real, indicando a necessidade de desenvolvimento e otimização de modelos mais complexos para este tipo de problema. Na Tabela 1 são apresentadas de forma resumida as *losses* e o tempo de treinamento de cada uma das abordagens propostas.

Tabela 1 – Comparando *loss* de treino e validação, e tempo de treinamento das abordagens testadas

	Loss	*Loss* de validação	Tempo de treinamento
RNA	64,77	4.851,21	142s
CNN	73,76	34.600,23	172s
RNN	267,82	1.349.339,88	355s

Fonte: elaborado pelo autor

A análise dos dados apresentados, referentes ao desempenho de diferentes arquiteturas de redes neurais (RNA, CNN e RNN) na modelagem de um sistema caótico, revela que nenhuma das abordagens testadas obteve resultados satisfatórios. O principal problema observado foi o *overfitting*, indicado pela alta *loss* de validação quando comparada à de treino (Erro Quadrático Médio - EQM), chegando em alguns casos, como o da RNN, próxima a 10^6. O *overfitting* ocorre quando o modelo aprende os dados de treinamento em detalhes excessivos, incluindo ruídos e flutuações aleatórias, o que compromete sua capacidade de generalizar para dados não vistos. Em outras palavras, o modelo "decora" os dados de treinamento em vez de aprender os padrões subjacentes, o que leva a um bom desempenho durante o treinamento, mas a previsões ruins em dados novos.

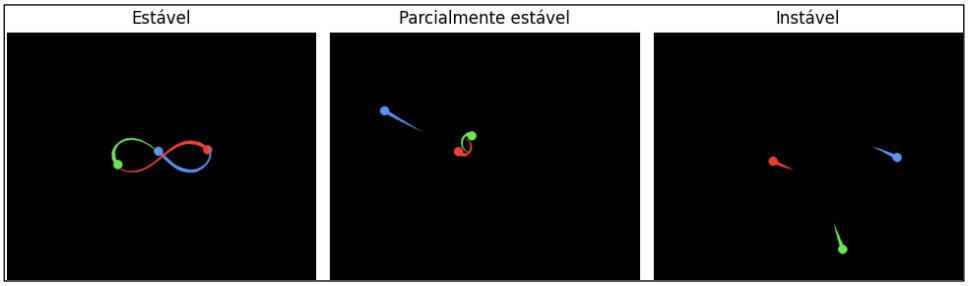
Conforme os dados da Tabela 1, embora a RNA tenha apresentado a menor *loss* (64,77), *loss* de validação (4.851,21) e tempo de treinamento (142s), é importante ressaltar que todas as *losses* são de ordens maiores ou iguais que 10^3, o que indica um desempenho ruim para todas as arquiteturas. A RNN, apesar de apresentar um tempo de treinamento significativamente maior (355s), não resultou em uma melhoria significativa em comparação às outras propostas. Dentre as possíveis causas para o *overfitting*, encontram-se: (i) Seleção do conjunto de dados: pouca variedade nos dados. Isso

pode impedir que os modelos aprendam padrões inerentes do problema, na maior parte do tempo aprendendo com dados em que a força gravitacional é muito próxima de zero. (ii) Arquitetura inadequada: A escolha da arquitetura da rede neural é crucial. É possível que as arquiteturas testadas não sejam complexas o suficiente para modelar o sistema caótico em questão. (iii) Hiperparâmetros inadequados: Os hiperparâmetros, como o otimizador e a taxa de aprendizado, influenciam diretamente no processo de treinamento. A escolha inadequada desses parâmetros pode levar ao *overfitting*.

3.3.2 MODELO SIMPLIFICADO

A partir dos resultados do modelo inicial, observou-se que o problema de três corpos poderia ser simplificado em três situações principais: sistema estável, parcialmente estável e instável. No primeiro caso os três corpos estão em total harmonia. No segundo, dois corpos interagem entre si enquanto o terceiro "voa" para longe. No último caso, o mais comum, os três corpos se dispersam. Esses casos são apresentados na Figura 16.

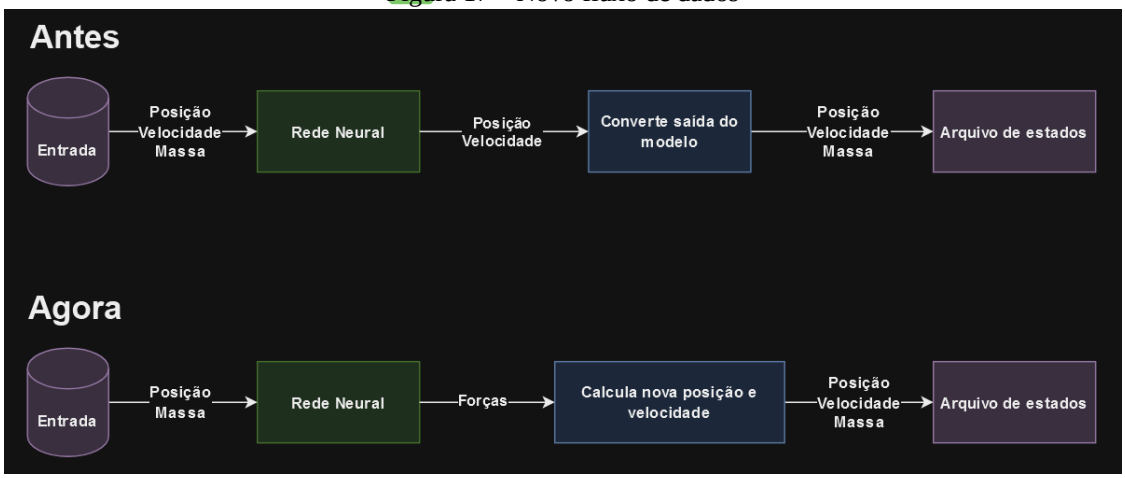
Figura 16 – Três situações principais no problema de três corpos



Fonte: elaborado pelo autor

Por consequência, a escolha do *dataset* ganha relevância, sendo crucial que ele contenha uma representação equilibrada de todos os cenários. Também se notou que um conjunto de dados mais enxuto favoreceu a generalização, sendo composto dessa vez por 900 transições de estado, separadas igualmente entre cada situação citada. Além disso, inicialmente esperava-se que o modelo entendesse a relação gravitacional e a convertesse para uma variação na velocidade e posição dos corpos. Essa abordagem foi simplificada, dividindo esses cálculos em duas etapas distintas. Primeiramente, o modelo recebe o estado inicial da simulação e retorna apenas o vetor de força aplicado a cada corpo. Em um segundo momento, essas forças são utilizadas para calcular as alterações nas demais propriedades dos corpos. A Figura 17 apresenta a diferença na forma como os dados serão trafegados nos modelos de aprendizado de máquina.

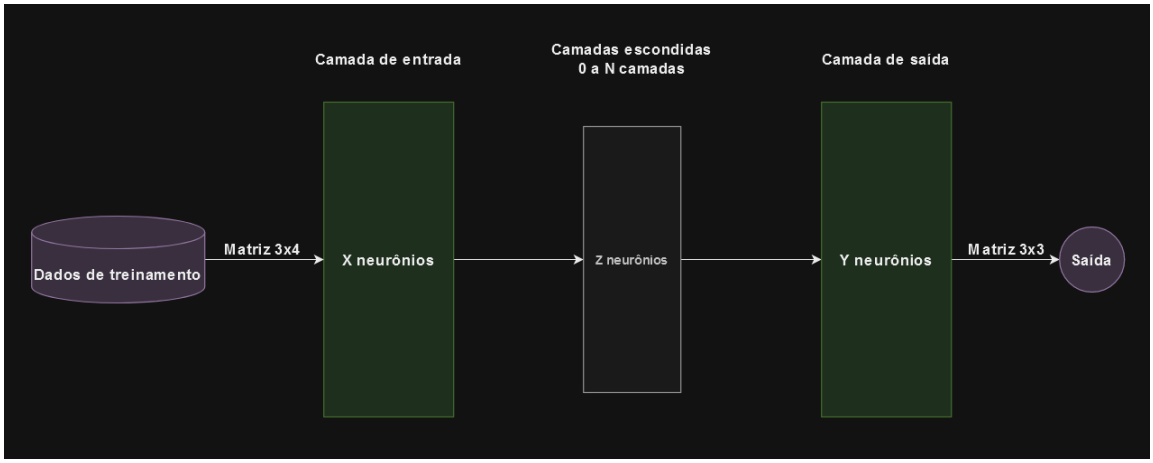
Figura 17 – Novo fluxo de dados



Fonte: elaborado pelo autor

Assim, agora a entrada do modelo é uma matriz 3x4, em que as propriedades dos corpos são apenas a posição (`x`, `y` e `z`) e a massa. A saída agora é uma matriz 3x3, contendo a força gravitacional aplicada à cada corpo (`x`, `y` e `z`). A Figura 18 ilustra a nova arquitetura.

Figura 18 – Arquitetura dos modelos



Fonte: elaborado pelo autor

O formato de entrada está ilustrado no Quadro 18, sendo que a primeira coluna descreve a estrutura da entrada e a segunda coluna apresenta um exemplo utilizando o estado inicial proposto por Chenciner & Montgomery (2000).

Quadro 18 – Entrada do modelo

[
  [pos_x₁, pos_y₁, pos_z₁, massa₁],
  [pos_x₂, pos_y₂, pos_z₂, massa₂],
  [pos_x₃, pos_y₃, pos_z₃, massa₃],
]
	[
	  [-0.97000436, 0.24308753, 0, 1],
	  [0, 0, 0, 1],
	  [0.97000436, -0.24308753, 0, 1],
	]

Estrutura da entrada	Estado inicial proposto por Chenciner & Montgomery (2000)

Fone: elaborado pelo autor

O formato de saída está especificado no Quadro 19. A primeira coluna descreve a estrutura da saída, enquanto a segunda coluna apresenta um exemplo real do retorno do modelo, baseado na entrada fornecida no Quadro 18.

Quadro 19 – Saída do modelo

[
  [forca_x₁, forca_y₁, forca_z₁],
  [forca_x₂, forca_y₂, forca_z₂],
  [forca_x₃, forca_y₃, forca_z₃],
]
	[
	  [1.212505, -0.303859, 0],
	  [0, 0, 0],
	  [-1.212505, 0.303859, 0],
	]

Estrutura de saída	Exemplo real retornado

Fonte: elaborado pelo autor

Nessa reestruturação, o algoritmo de conversão da saída do modelo foi alterado. Agora, ele aplica cálculos similares ao do simulador para aplicar a mudança na velocidade e posição de cada um dos corpos. O Quadro 20 detalha esse processo.

Quadro 20 – Algoritmo de conversão da saída do modelo para o arquivo de estados

```
estado_inicial = carregar_estado_inicial()
novo_estado = []
para i, forca em saída_modelo:
    corpo = estado_inicial[i]
    aceleração = forca / corpo.massa
    velocidade = corpo.velocidade + aceleração * time_step
    posição = corpo.posição + corpo.velocidade * time_step
    novo_estado.adicionar({
        "pos": posição,
        "vel": velocidade,
        "mass": corpo.mass
    })
```

Fonte: elaborado pelo autor

Diante dessa modificação, foram realizados novos testes com a RNA, CNN e RNN. A arquitetura interna permaneceu similar à descrita na seção 3.3.1. Entretanto, observou-se que a modificação na arquitetura das redes neurais, com o uso de camadas internas de 128 nós, resultou em uma melhora substancial na capacidade de modelagem do sistema caótico. A análise comparativa demonstrou que a RNA, apesar de ter reduzido sua *loss* significativamente, ainda ficou

aquém da CNN e da RNN. A CNN, por sua vez, se destacou com a menor *loss*, *loss* de validação e o menor tempo de treinamento, indicando uma capacidade superior de generalização e aprendizado dos padrões presentes nos dados. Essa maior eficiência da CNN pode ser atribuída à sua arquitetura, que, por meio de suas camadas convolucionais, consegue extrair características espaciais relevantes para a representação do sistema caótico. Detalhes sobre as *losses* e o tempo de treinamento de cada técnica são apresentados na Tabela 2.

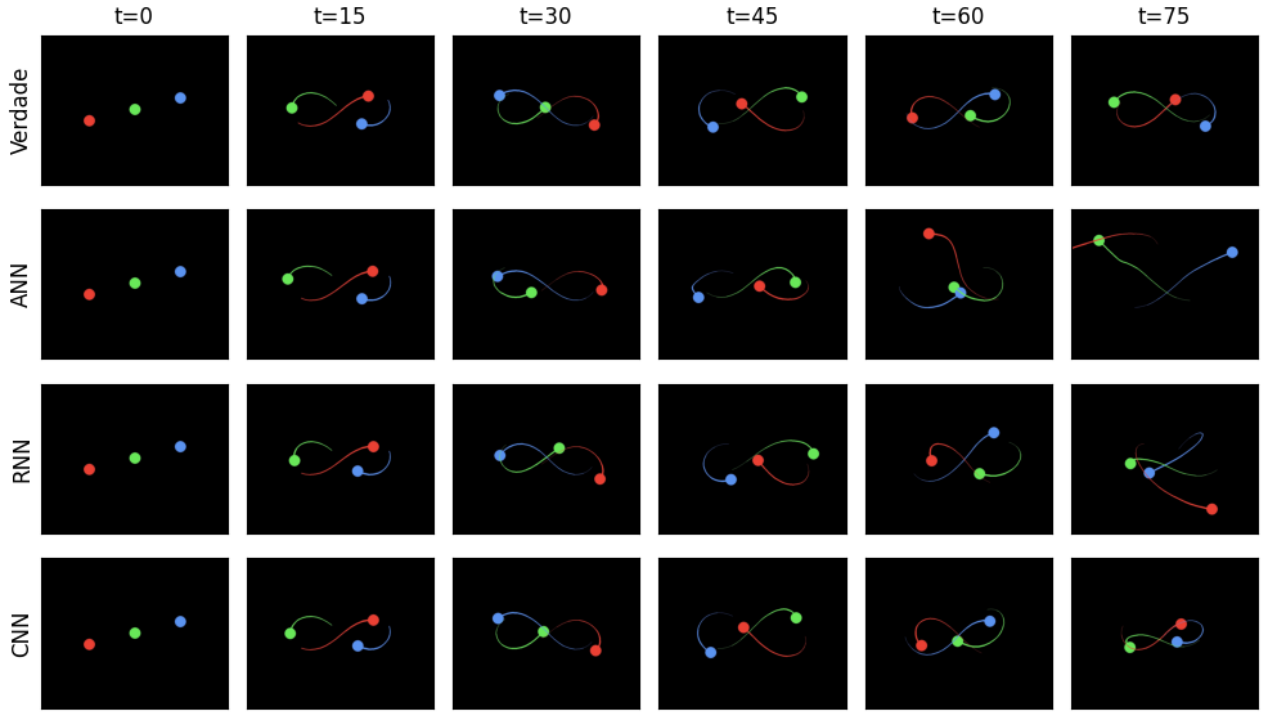
Tabela 2 – Comparando *loss* e tempo de treinamento das abordagens testadas

	Loss	*Loss* de validação	Tempo de treinamento
RNA	0,00527	0,13496	158s
CNN	0,00017	0,00020	144s
RNN	0,00084	0,00067	171s

Fonte: elaborado pelo autor

A partir dos treinos, avaliou-se o efeito caótico sobre cada uma delas. Para isso, foi utilizada como entrada a simulação com formato de oito proposta por Chenciner e Montgomery (2000). Em seguida, a simulação foi aplicada recursivamente a cada um dos modelos. A Figura 19 compara as três abordagens com a simulação exata em seis momentos diferentes.

Figura 19 – Comparação do efeito caótico sobre a ANN, RNN e CNN



Fonte: elaborado pelo autor

A partir da Figura 19 pode-se observar o desempenho de três arquiteturas de redes neurais diferentes (ANN, RNN e CNN) na previsão das trajetórias de três corpos ao longo do tempo, em comparação com a verdade fundamental. Diante das simulações dos efeitos caóticos, pode se fazer as seguintes análises:

a) Verdade fundamental (Verdade): esta linha descreve o movimento real dos três corpos (representados por pontos vermelhos, verdes e azuis) ao longo do tempo (t=0 a t=75). Os corpos seguem um padrão distinto em forma de oito.
b) Rede Neural Artificial (ANN): a ANN tem dificuldades em prever com precisão o movimento, particularmente após os passos iniciais de tempo. As trajetórias previstas desviam-se significativamente dos caminhos reais, destacando uma compreensão deficiente da dinâmica subjacente.
c) Rede Neural Recorrente (RNN): a RNN demonstra uma ligeira melhoria em relação à ANN. Embora capture melhor o padrão geral de movimento, as trajetórias previstas ainda divergem da verdade fundamental ao longo do tempo, indicando limitações na captura das complexidades do sistema.
d) Rede Neural Convolucional (CNN): a CNN exibe as previsões mais precisas, espelhando de perto as trajetórias da verdade fundamental ao longo de toda a sequência temporal. Isso sugere que a CNN aprendeu efetivamente as relações espaciais e dependências temporais dentro dos dados, permitindo que ela generalize bem e faça previsões precisas.

Contudo, pode-se perceber visualmente o desempenho superior da CNN na modelagem e previsão do movimento complexo e caótico dos três corpos. As trajetórias geradas pela CNN se mantêm fiéis ao padrão em forma de oito da "Verdade", mesmo para tempos mais avançados. Em contraste, a RNA, com a maior perda, apresenta os desvios mais gritantes, perdendo a trajetória correta logo nos estágios iniciais. A RNN, embora supere a RNA, ainda demonstra dificuldades em manter a precisão ao longo do tempo, confirmando a análise da tabela de *loss* (Tabela 2). Portanto, a concordância entre a tabela de desempenho, que quantifica a *loss* de cada modelo, e a representação visual das trajetórias previstas, solidifica a conclusão de que a CNN, com sua capacidade de extrair e interpretar padrões espaço-temporais complexos, se destaca como a arquitetura mais eficaz para modelar o sistema caótico em questão.

4 CONCLUSÕES

Este estudo abordou o desafio do problema dos três corpos utilizando redes neurais como base. Sua relevância se estende a pesquisas sobre evolução de sistemas cósmicos, formações de sistemas planetários, galáxias, entre outros campos. Sua influência se expande a outras áreas tal como a dinâmica de fluídos e a biologia (RUCCI *et al*., 2020).

O objetivo central foi desenvolver um modelo de inteligência artificial capaz de prever a força gravitacional aplicada a cada corpo. Para isso, foram exploradas as capacidades de RNAs, CNNs, RNNs e algoritmos de aprendizado por reforço para realizar essas previsões. Também buscou-se entender quão bem eles poderiam lidar com a natureza caótica do problema. Como suporte, foram desenvolvidas duas ferramentas: o simulador de N corpos e o visualizador. O simulador recebe um estado inicial de simulação e parâmetros relevantes, gerando dois arquivos: um com a evolução das posições e velocidades dos corpos, e outro com as forças aplicadas a cada um deles. O visualizador processa o arquivo de saída do simulador, proporcionando uma visualização em tempo real da simulação e salvando imagens de cada *frame*.

Com a apuração dos resultados, observou-se a CNN foi a arquitetura que melhor desempenhou na abstração da natureza caótica do problema dos três corpos, superando RNAs e RNNs. O aprendizado por reforço não se mostrou uma solução muita apropriada, visto que grande parte dos algoritmos trabalham com um mapa de estados discreto, tornando necessário o uso de redes neurais para torná-los contínuos. Por ser redundante aos outros estudos, o aprofundamento nessa técnica foi descartado. Além disso, o uso de um simulador de N corpos e de uma ferramenta de visualização se mostraram eficazes para a construção e avaliação dos modelos. O primeiro permitiu maleabilidade para construção da base de dados, facilitando o seu balanceamento. O segundo permitiu uma avaliação intuitiva dos efeitos caóticos sobre as redes neurais, fato que trouxe diversos *insights* durante o desenvolvimento.

Em comparação aos trabalhos correlatos, como o de Breen *et al. (2020)*, que utilizou RNAs para prever trajetórias em encontros próximos, o trabalho se destacou pela exploração adicional de CNNs e RNNs. Inclusive, a limitação de corpos precisarem possuir a mesma massa também foi removida. Gonzalez *et al. (2020)*, utilizou redes de grafos aplicadas a simulações de materiais, enquanto nossa abordagem focou diretamente em simulações gravitacionais. Por fim, Oliveira et al. (2020) usaram CNNs para simulações cosmológicas, complementando nossa análise com uma abordagem similar, porém em um contexto diferente.

As limitações encontradas incluem dificuldades em generalizar os modelos para interações com mais de três corpos e a persistência do efeito caótico em simulações prolongadas. Para mitigar essas limitações, poder-se-iam investigar mais profundamente o impacto da seleção de dados na construção de modelo baseados em física. Isso envolve entender se casos extremos devem ser tratados de forma equivalente aos demais dados ou como *outliers*. Ademais, estudar diferentes abordagens acerca do problema, buscando interpretações que não lidem diretamente corpo a corpo, mas que tratem o sistema de forma unificada.

Para concluir, entende-se que a pesquisa cumpriu seus objetivos ao apresentar a criação de um modelo capaz de entender a relação complexa e caótica inerente ao problema de três corpos. Com a ajuda de ferramentas que se provaram úteis no processo, o simulador e o visualizador, notou-se pontos de melhoria no enfrentamento do problema. Com isso, o estudo permitiu um avanço do campo da inteligência artificial aplicada à física, pois não aplicou limitações relacionadas à massa dos corpos. Espera-se que este estudo possa inspirar cientistas a se aprofundarem nessa área, de modo a criar um caminho para o desenvolvimento de novas tecnologias.

5 REFERÊNCIAS

ALVES-BRITO, Alan, CORTESI, Arianna. Complexidade em Astronomia e Astrofísica. Revista Brasileira de Ensino de Física, v. 43, n. suppl 1, 2021.

ANGULO, Raul E.; WHITE, Simon D. M. One simulation to fit them all - changing the background parameters of a cosmological N-body simulation. Monthly Notices of the Royal Astronomical Society, abr. 2010.

BODE, Paul; OSTRIKER, Jeremiah P.; XU, Guohong. The Tree Particle-Mesh N -Body Gravity Solver. The Astrophysical Journal Supplement Series, v. 128, n. 2, p. 561–569, jun. 2000.

BREEN, Philip G. et al. Newton versus the machine: solving the chaotic three-body problem using deep neural networks. Monthly Notices of the Royal Astronomical Society, Oxford, v. 494, n. 2, p. 2465–2470, 22 abr. 2020.

BURTSCHER, Martin; PINGALI, Keshav. An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm. Elsevier eBooks, p. 75–92, 1 jan. 2011.

CHENCINER, A.; MONTGOMERY, R. A Remarkable Periodic Solution of the Three-Body Problem in the Case of Equal Masses. The Annals of Mathematics, v. 152, n. 3, p. 881, nov. 2000.

CORRÊA, Débora Cristina. Sistema baseado em redes neurais para composição musical assistida por computador. 2008. 165 f. Dissertação (Mestrado em Ciências Exatas e da Terra) - Universidade Federal de São Carlos, São Carlos.

COSTA, Felipe et al. Automatic Generation of Natural Language Explanations. 5 mar. 2018.

D'ARBO JUNIOR, Hélio. Redes neurais recorrentes para produção de sequências temporais. 1998. 139 f. Dissertação (Mestrado em Engenharia Elétrica) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos.

DENG, Li; YU, Dong. Deep Learning Methods and Applications. Foundations and Trends in Signal Processing, Hanover, v. 7, n. 3-4, p. 197-387, jun. 2014.

FUJIWARA, Yoshi. Visualizing a Large-Scale Structure of Production Network by N-Body Simulation. Progress of Theoretical Physics Supplement, v. 179, p. 167–177, 2009.

GARRISON, Lehman H. et al. The abacus cosmological N-body code. Monthly Notices of the Royal Astronomical Society, Oxford, v. 508, n. 1, p. 575–596, 7 set. 2021.

GIERSZ, Mirek; HEGGIE, Douglas C. Statistics of N-body simulations -- III. Unequal masses. Monthly Notices of the Royal Astronomical Society, v. 279, n. 3, p. 1037–1056, 1 abr. 1996.

GOMES, Daniel Takata. Modelos de redes neurais recorrentes para previsão de series temporais de memorias curta e longa. 2005. 137 f. Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matematica, Estatistica e Computação Cientifica, Campinas.

GONZALEZ-SANCHEZ, Alvaro et al. Learning to Simulate Complex Physics with Graph Networks. Ithaca, [2020]. Disponível em: https://arxiv.org/abs/2002.09405. Acesso em: 23 jun. 2024.

HAYKIN, Simon. Redes neurais: princípios e prática. Porto Alegre: Bookman, 2001.

HERNANDEZ, D. M. Improving the accuracy of simulated chaotic N-body orbits using smoothness. Monthly Notices of the Royal Astronomical Society, Oxford, v. 490, n. 3, p. 4175–4182, 27 set. 2019.

HORCHEITER, Sepp; SCHMIDHUBER, Jurgen. Long Short-Term Memory. Neural computation, Cambridge, v. 9, n. 8, p. 1735-1780, nov. 1997.

HUT, Piet; MAKINO, Junichiro. The N-Body Problem: From Leapfrog to Runge-Kutta, 2007.

IGUAL, Laura; SEGUÍ, Santi. Introduction to Data Science. Introduction to Data Science: A Python Approach to Concepts, Techniques and Applications, 1 jan. 2024.

JAIN, Pushpendra K.; NKOMA, John S. Introduction to Classical Mechanics: Kinematics, Newtonian and Lagrangian. Mkuki na Nyota Publishers, 2019.

KLEIN, Carlos; MARTINS, Joel. Implementação de rede neural em hardware de ponto fixo. Brasília, 2006. <http://bdm.unb.br/bitstream/10483/915/1/2006_JoelMartins_CarlosKlein.pdf>. Acesso em: 21 de jun. de 2024.

LI, Zewen et al. A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects. IEEE Transactions on Neural Networks and Learning Systems, v. 33, n. 12, p. 1–21, 2021.

MACIEL, Josias. Análise de um sistema de crédito cooperativo através de redes neurais (MLP) com a utilização do algoritmo levenberg marquardt. 2005. 87 f. Dissertação (Mestrado em Ciências) – Curso de Pós-graduação em Métodos Numéricos em Engenharia, Universidade Federal do Paraná, Curitiba.

MCLAUGHLIN, William. N-Body visualizations. Thesis. Rochester Institute of Technology. New York, 2006. <https://repository.rit.edu/theses/6912>. Acesso em: 21/06/2024.

MUKHERJEE, Diptajyoti et al. Fast Multipole Methods for N-body Simulations of Collisional Star Systems. The Astrophysical Journal, v. 916, n. 1, p. 9, 1 jul. 2021.

MUSIELAK, Zdzislaw E.; QUARLES, Billy. The three-body problem. Reports on Progress in Physics, v. 77, n. 6, p. 065901, 2014.

NYLONS, Lars; HARRIS, Mark; PRINS, Jan. Fast n-body simulation with cuda. GPU gems, v. 3, p. 62-66, 2007.

OLIVEIRA, Renan Alves de et al. Fast and Accurate Non-Linear Predictions of Universes with Deep Learning. arXiv (Cornell University), 2 dez. 2020.

PRIGARIN, Vladimir; KARAVAEV, Dmitry; PROTASOV, Viktor. The cuFFT code for N-body simulation. Journal of physics. Conference series, v. 1336, n. 1, p. 012023–012023, 1 nov. 2019.

QUINN, Thomas et al. Time stepping N-body simulations. arXiv (Cornell University), 1 jan. 1997.

RIBEIRO, Edivaine G. Rede neural convolucional aplicada ao reconhecimento de passagens de nível clandestinas em Ferrovias. 2020. 59 f. Monografia (Especialista em Sistemas Inteligentes Aplicados a Automação) - Instituto Federal do Espírito Santo, Vitória, 2020.

ROSA, Renan P. Método de classificação de pragas por meio de rede neural convolucional profunda. 2018. 101 f. Dissertação (Mestrado em Computação Aplicada) - Universidade Estadual de Ponta Grossa, Ponta Grossa, 2018.

ROY, Nikhil Ranjan. Introduction to classical mechanics. Vikas Publishing House, 1990.

RUCCI, Enzo et al. Optimization of the N-Body Simulation on Intel's Architectures Based on AVX-512 Instruction Set. Communications in computer and information science, p. 37–52, 1 jan. 2020.

SALAZAR, Andrés Eduardo Coca. Mineração de estruturas musicais e composição automática utilizando redes complexas. 2015. 197 f. Tese (Doutorado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos.

SANTANA, Luciana Maiara Queiroz de. Aplicação de redes neurais recorrentes no reconhecimento automático da fala em ambientes com ruídos. 2017. 68 f. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal de Sergipe, São Cristóvão.

SOARES, Gustavo R.; CARMO, Alisson F. C. Viabilidade de detecção de câncer de mama através de rede neural convolucional em mamografias. ETIC-ENCONTRO DE INICIAÇÃO CIENTÍFICA-ISSN 21-76-8498, [S. l.], v. 16, n. 16, p. 1 - 15, set. 2020.

SPURZEM, Rainer. Direct N-body simulations. Journal of Computational and Applied Mathematics, v. 109, n. 1-2, p. 407–432, set. 1999.

WANG, Long et al. petar: a high-performance N-body code for modelling massive collisional stellar systems. Monthly Notices of the Royal Astronomical Society, Oxford, v. 497, n. 1, p. 536–555, 24 jul. 2020.

WU, Stephen G. et al. A leaf recognition algorithm for plant classification using probabilistic neural network. In: IEEE INTERNATIONAL SYMPOSIUM ON SIGNAL PROCESSING AND INFORMATION TECHNOLOGY, 7th, 2007, Giza. Proceedings… [S.l]: IEEE, 2008. p. 11-16.