

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

Alex Souza Carvalho

Métodos de execução paralela em GPU visando o aprendizado de máquina em
programas de docking molecular proteína-ligante

São Carlos
2024

Alex Souza Carvalho

Métodos de execução paralela em GPU visando o aprendizado de máquina
em programas de docking molecular proteína-ligante

Trabalho de conclusão de curso apresentado ao
Instituto de Física de São Carlos da Universidade de
São Paulo para obtenção do título de Bacharel em
Ciências Físicas e Biomoleculares.

Orientador: Prof. Dr. Alessandro Silva Nascimento -.

São Carlos
2024

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Carvalho, Alex Souza

Métodos de execução paralela em GPU visando o aprendizado de máquina em programas de docking molecular proteína-ligante / Alex Souza Carvalho; orientador Alessandro Silva Nascimento -- São Carlos, 2024.

26 p.

Trabalho de Conclusão de Curso (Bacharel em Ciências Físicas e Biomoleculares) -- Instituto de Física de São Carlos, Universidade de São Paulo, 2024.

1. Docking molecular. 2. Biologia computacional estrutural. 3. Aprendizado de máquinas. I. Nascimento, Alessandro Silva, orient. II. Título.

RESUMO

Com as inovações na área de inteligência artificial, experienciaram-se diversas inovações na área de biologia computacional, em particular na biologia estrutural. Mediante essas novas tecnologias, torna-se imediatamente claro que a coleção de dados biológicos sendo catalogados há décadas podem ser utilizados para alimentar sistemas de aprendizado de máquina (ou *machine learning*) e potencialmente aprimorar as ferramentas clássicas de análise biomolecular, visando a engenharia de proteínas ou as interações proteína-proteína e proteína-ligante. É sob esta última perspectiva que se apresenta este trabalho destinado à extensão do software pyLiBELA, desenvolvido no Instituto de Física de São Carlos, e que implementa um algoritmo para o *docking* de ligantes baseado em campos de força AMBER. Utilizando-se do *multithreading* em placas gráficas da NVIDIA, por meio da ferramenta CUDA, obtiveram-se substanciais ganhos de performance na etapa crucial de computação da representação espacial, em *grids*, dos potenciais devidos à proteína, demonstrando a eficácia do método para qualquer sistema de *docking* que utilize uma representação similar. Tomando proveito deste ganho, ingressa-se na etapa de construção de redes neurais convolutivas treinadas nas representações de *grids* dos complexos proteínas-ligantes do conjunto de dados do PDBind, a fim de obter-se um modelo que generaliza uma pontuação da afinidade do complexo. Os resultados obtidos com modelos convolucionais demonstraram uma capacidade de aprendizado do modelo, revelando um potencial deste tipo de representação para a análise molecular da interação proteína-ligante.

Palavras-chave: Docking molecular. Biologia computacional estrutural. Aprendizado de máquinas

SUMÁRIO

1 INTRODUÇÃO.....	7
2 MATERIAIS E METODOS.....	11
2.1 Implementação em Numba.....	11
2.2 Implementação em CUDA.....	11
2.3 Aprendizado de máquina.....	13
3 RESULTADOS.....	17
3.1 Performance em CUDA.....	17
3.2 Resultados do aprendizado de máquina.....	18
3.2.1 Modelo CNN – TensorFlow.....	19
4 CONCLUSÕES E CONSIDERAÇÕES FINAIS.....	23
REFERÊNCIAS.....	25

1 INTRODUÇÃO

Ferramentas computacionais têm sido, no decorrer das últimas décadas, cada vez mais utilizadas em diferentes áreas pertinentes à biologia. De particular interesse ao planejamento de fármacos, a introdução do chamado *computer aided drug design* (CADD) permite triar, *in silico*, um grande número de compostos disponíveis entre diversos bancos de dados, aliviando o custo e tempo de experimentação laboratorial, através da significativa redução do espaço de compostos candidatos (1). *Ligand-based drug design* (LBDD) e *structure-based drug design* (SBDD) são as duas metodologias pertinentes ao CADD. O LBDD parte de informações dos ligantes conhecidos para um alvo molecular de interesse e visa traçar a relação entre as características químicas e sua atividade para que se possa, assim, tentar otimizar as suas propriedades farmacológicas. Já a estratégia de SBDD parte de informações estruturais tridimensionais da macromolécula de interesse com a finalidade de se extrair informações dos sítios de interação relevantes à função biológica, permitindo-se então derivar e testar novos ligantes que atuem sobre esses sítios (2).

Dentre os métodos empregados em SBDD, o *docking* molecular é o mais utilizado. O objetivo do *docking* molecular é avaliar e prever a interação de um complexo ligante-receptor, usualmente de um ligante a uma proteína no contexto da biologia molecular. O funcionamento de softwares de *docking* dependem, em geral, de duas etapas: um algoritmo que gera uma amostragem de diferentes conformações possíveis do ligante em relação à macromolécula seguido de uma função de pontuação para avaliar e ranquear a interação do complexo ligante-proteína gerado pelo confôrmero da etapa anterior (3). Essas duas etapas são reiteradas até achar-se uma conformação do ligante que otimiza a pontuação.

O pyLiBELa é um programa de *docking* molecular em ativo desenvolvimento no Instituto de Física de São Carlos, estendendo a funcionalidade do software antecessor LiBELa, desenvolvido pelo grupo do prof. Alessandro Nascimento e com o mesmo instituto (4). O programa LiBELa (acrônimo para *Ligand Binding Energy Landscape*), desenvolvido desde 2012, emprega um método híbrido em que a amostragem de conformações do ligante é orientada de início pela sobreposição do ligante de interesse a um ligante de referência (que já esteja em sua conformação tridimensional no complexo de interesse). Essa sobreposição é feita através da descrição gaussiana de sua forma e distribuição de cargas, seguindo o algoritmo regente do programa MolShaCS (4-5). Uma vez determinada a conformação inicial do ligante de interesse, novos confôrmeros são gerados através do algoritmo genético (*genetic algorithm*) da biblioteca de ferramentas computacionais químicas do

OpenBabel (6). Os confôrmeros são, então, pontuados segundo uma função baseada em campos de força AMBER (7). A energia de interação é definida tipicamente em um potencial amortecido dado

por:

$$E = \sum_i \sum_j \left[\frac{A_{ij}}{(r_{ij}^6 + \delta^6)^2} - \frac{B_{ij}}{(r_{ij}^6 + \delta^6)} + \frac{332.0 q_i q_j}{D(r_{ij}^6 + \delta_{es}^6)^{1/3}} \right] + (E^{sol}) + (E^{HB})$$

No somatório sobre os átomos i e j do ligante e do receptor, respectivamente, os dois primeiros termos representam as interações de van der Waals através de um potencial Lennard-Jones 12-6, onde os numeradores derivam de uma média geométrica

$$A_{ij} = 2^{12} \sqrt{\epsilon_i \epsilon_j} (r_i r_j)^6 \mathbf{e} \quad B_{ij} = 2^7 \sqrt{\epsilon_i \epsilon_j} (r_i r_j)^3 \quad (2)$$

com ϵ e r representando parâmetros do campo de força AMBER original [8]. Já o terceiro termo da Equação 1 avalia as interações eletrostáticas entre as cargas q_i e q_j dos átomos do ligante e receptor por meio de um potencial de Coulomb. Esses três primeiros termos, baseados nas parcelas intermoleculares da função AMBER, possuem um termo de amortecimento (*smoothing*) δ e δ_{es} , ambos dados como 2.5 Å [9, 10]. E^{sol} é um termo de dessolvatação, formado pela combinação de dois fatores: um que conta a afinidade entre um átomo e o solvente como sendo linear em relação ao quadrado da carga atômica:

$$S_i = \alpha q_i^2 + \beta$$

com $\alpha = 0.1$ kcal/(mol·e²) e $\beta = -0.005$ kcal/mol. E o segundo fator dado pelo volume de solvente deslocado na interação, dado por um envelope Gaussiano:

$$X_j = \left(\frac{4\pi}{3\sigma^3} \right) a^3 \exp\left(\frac{-r_{ij}^2}{2\sigma^2} \right) \quad (4)$$

onde r é a distância interatômica entre i e j , a é o raio efetivo do átomo e $\sigma = 3.5$ Å é uma constante (8). A energia de solvatação é dada, então, por:

$$E^{sol} = \sum_i \sum_j S_i X_j + S_j X_i$$

Finalmente, o último termo da Equação 1 modela as ligações de hidrogênio por meio de um potencial 10-12 e um termo de direcionalidade $\cos^4(\theta)$, onde θ é o ângulo entre o doador, átomo de hidrogênio e aceptor (10).

Neste método de cálculo de energia, o custo computacional aumenta com o número de átomos interagentes, e recalcular a energia total após cada pesquisa conformacional do ligante se torna rapidamente limitante (11). Para aliviar este custo, pode-se, por exemplo, utilizar os termos separáveis da Equação 1:

$$E = \sum_{i=1}^{\text{lig}} \left[\sqrt{\epsilon_i} r_i^6 \sum_{j=1}^{\text{rec}} \frac{2^{12} \sqrt{\epsilon_j} r_j^6}{(r_{ij}^6 + \delta^6)^2} - \sqrt{\epsilon_i} r_i^3 \sum_{j=1}^{\text{rec}} \frac{2^7 \sqrt{\epsilon_j} r_j^3}{(r_{ij}^6 + \delta^6)} + q_i \sum_{j=1}^{\text{rec}} \frac{332.0 q_j}{D(r_{ij}^6 + \delta_{es}^6)^{1/3}} \right] \quad (6)$$

e similarmente para E^{sol} e E^{HB} . Em seguida, utilizando uma rede de pontos uniformemente espaçados delimitando o espaço de pesquisa conformacional do ligante (i.e., um *grid*), pode-se pré-calcular, em cada um dos pontos, os termos que são somas sobre os átomos do receptor (9). Obtém-se, assim, um *grid* de valores separáveis como:

$$\text{vdwA} = \sum_{j=1}^{\text{rec}} \frac{2^{12} \sqrt{\epsilon_j} r_j^6}{(r_{ij}^6 + \delta^6)^2}, \quad \text{vdwB} = \sum_{j=1}^{\text{rec}} \frac{2^7 \sqrt{\epsilon_j} r_j^3}{(r_{ij}^6 + \delta^6)} \quad \mathbf{e} \quad \text{elec} = \sum_{j=1}^{\text{rec}} \frac{332.0 q_j}{D(r_{ij}^6 + \delta_{es}^6)^{1/3}} \quad (7)$$

que podem rapidamente ser integrados no cálculo da energia de um conformero contido no *grid*, tomando a posição de cada átomo do ligante e realizando uma interpolação trilinear dos oito pontos do *grid* mais próximos, estimando, assim, o fator da energia devido ao receptor naquele átomo.

Apesar de benéfico ao longo de diversas pesquisas conformacionais para um dado receptor, o custo inicial de calcular o *grid* se torna oneroso quando se busca realizar um estudo sobre um grande volume de dados de complexos receptores-ligantes. Para endereçar esse problema, foram empregados métodos computacionais de execução paralela em placa de vídeo (GPU, do inglês *graphics processing unit*) para otimizar o tempo de computação, aplicado ao pyLiBELa.

O projeto pyLiBELa estende a API do LiBELa expondo seus módulos, escritos originalmente em linguagem C++, para a linguagem Python, de forma a permitir a utilização do programa em serviços da nuvem como o Google Colaboratory (ou Colab). Como descrito, o algoritmo subjacente de *docking* molecular do pyLiBELa envolve um sistema especialista clássico, porém novos avanços na área de biologia estrutural computacional através de aprendizado de máquina (ML, do inglês *machine learning*) introduzem um novo paradigma para abordar estes problemas. Novos sistemas são capazes de expandir a base de dados e o conhecimento sobre estruturas de moléculas biológicas, como o AlphaFold2 (12) e o RoseTTAFold (13), além de prospectivas funcionalidades de predição estrutural para a avaliação da interação de biomoléculas (14). Para criar-se potenciais soluções de ML nesta área, requer-se a aplicação adequada de um

modelo e representação do espaço problema (no caso, o espaço físico-químico das interações intermoleculares) (15).

Neste escopo, uma hipótese que tem sido levantada no grupo de pesquisa do prof. Alessandro Nascimento é: a representação de uma macromolécula biológica na forma dos seus potenciais de interação descritos em um conjunto de *grids* pode ser usada para o *docking* de ligantes de forma mais rápida e mais eficiente?

Sabemos que os *grids* são pontos discretos contendo todas as informações de interação (Equação 7) e que formam algo análogo a uma imagem tridimensional. Assim, buscando responder a esta hipótese, utilizou-se os *grids* gerados pela versão CUDA do programa pyLIBELA para o treinamento de um modelo preliminar de rede neural convolucional (ou CNN, do inglês *convolutional neural network*), considerando a eficácia deste tipo de modelo no reconhecimento de dados de imagens (16), a fim de gerar inferências sobre a interação de pares ligante-proteína. O pyLiBELa é um projeto código aberto e as contribuições pertinentes ao trabalho estão disponíveis no repositório do [GitHub](#).

Desta forma, este trabalho tem dois objetivos principais: (i) a implementação do cálculo dos *grids* de interação em placas gráficas (GPU) e; (ii) o emprego destes *grids* para um conjunto experimental de dados de proteína-ligante no treinamento de um modelo CNN *deep learning* visando o ranqueamento ou a pontuação das interações proteína-ligante.

2 MATERIAIS E MÉTODOS

2.1 Implementação em Numba

Para realizar a otimização do processamento dos *grids*, inicialmente utilizou-se a ferramenta Numba diretamente sobre os módulos do pyLiBELa expostos em Python através da biblioteca Boost.Python em C++ (17). Numba é um compilador *just-in-time* (JIT) para CPython que visa utilizar as representações de dados de *ndarrays* nativas ao Numpy (18), que são altamente otimizadas, porém aprimorando semânticas comuns do código que precisam ser delegados ao interpretador Python (que é comparativamente lento), como indexação de arrays e loops. Com o Numba, o usuário pode marcar as funções que deverão ser JIT-compiladas durante o *runtime* e também requisitar a execução do código em GPU (19).

Utilizando o ambiente do Google Colab com acesso a uma GPU NVIDIA T4, as funções do LiBELa responsáveis pelos cálculos dos *grids* foram reescritos diretamente em Python adaptando as variáveis e os arrays necessários, expostos pelo Boost.Python, em *ndarrays* de Numpy, preservando a mesma lógica e marcando as funções para otimização em Numba e execução em GPU. Este método foi logo abandonado quando se notou tempos de execução em ordem de magnitudes maiores em relação a simplesmente invocar a função original do cálculo do *grid*, em C++. Isso deveu-se ao fato do Numba não conseguir interpretar e otimizar os objetos gerados pela interface do Boost.Python, acarretando na inexorável circunstância de que todo o volume de dados oriundo do *pipeline* do pyLiBELa antes da geração dos *grids* precisa ser convertido em *ndarray* e, depois de passar pelas funções reescritas, serem convertidas novamente para o objeto do Boost.Python. Ambas as etapas envolvem enormes transferências de dados que precisam ser feitas pelo interpretador Python, tornando a solução proibitivamente ineficiente.

2.2 Implementação em CUDA

Para endereçar o problema, decidiu-se então empregar o uso de linguagem CUDA C, embutindo a lógica de paralelização juntamente ao código fonte do LiBELa. CUDA, ou *Compute Unified Device Architecture*, se refere à arquitetura introduzida desde 2006 em placas de vídeo da NVIDIA, dotando suas unidades de processamento com um conjunto de instruções e acesso arbitrário de memória, tornando-as propícias para a realização de computação generalizada, onde

uma vez eram restritas apenas às funcionalidades domínio-específico da computação gráfica (20). O benefício dessa arquitetura está na capacidade da placa gráfica de realizar execução concorrente em milhares de *threads* distribuídos entre até centenas de núcleos de processamento (para dispositivos modernos). A programação paralela é facilitada pelo CUDA C/C++, uma extensão das linguagens sequenciais C e C++ que, com algumas adições, oferece uma API simples para coordenar a execução concorrente nos *threads* (21).

Como o código CUDA é capaz de se integrar com C++ nativo, implementar paralelismo ao *pipeline* do pyLiBELa é relativamente direto. Figura 1 ilustra as etapas de execução envolvidas.

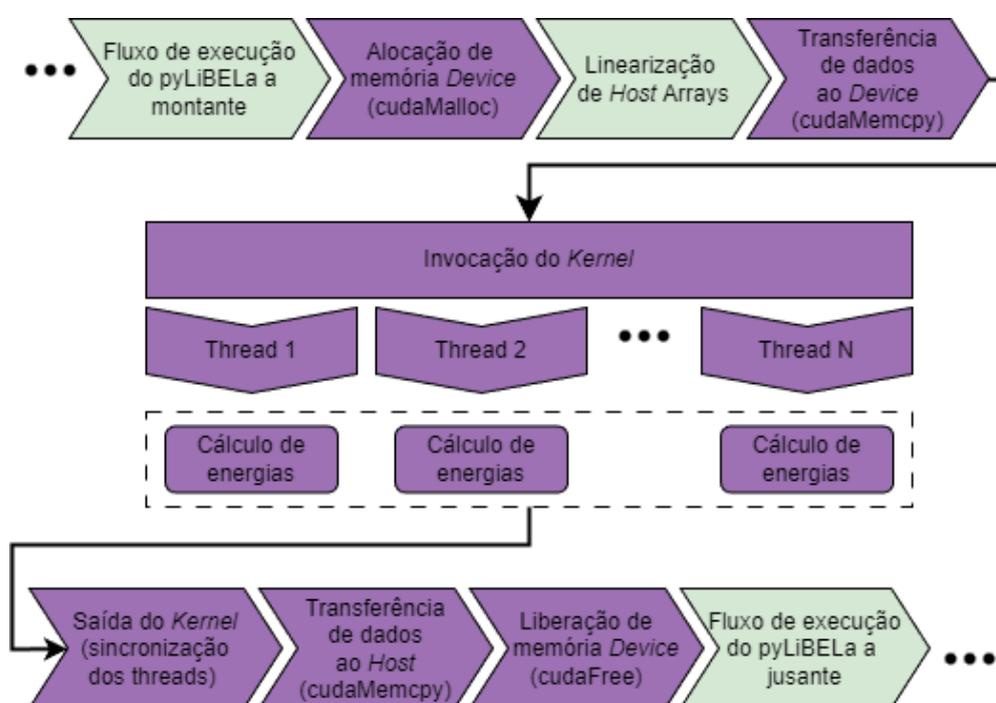


Figura 1 - Fluxo de execução dos cálculos dos *grids* utilizando CUDA. Representados em verde, etapas da execução pertinentes ao *host*¹. Representados em roxo, etapas pertinentes ao *device*.

Fonte: Compilado pelo autor.

Após algumas rotinas preparatórias e o processamento dos arquivos do ligante, receptor e ligante de referência, geralmente em formato SYBYL (**.mol2**), com as configurações definidas pelo usuário, o programa inicia as funções de cálculo dos *grids*. Para adaptar as representações de dados originais ao uso de CUDA, toda a informação contida em objetos **std::vector** de C++ precisa ser transferida para *arrays* em estilo C, além de linearizados, no caso dos vetores multidimensionais,

¹ No contexto de programação em CUDA, *host* se refere à CPU e *device* à GPU (20).

para que se conformem contiguamente em memória (um requerimento para a operação de cópia subsequente). Em seguida, toda a memória necessária a ser usada na GPU precisa ser alocada (através da função `cudaMalloc`) e os dados previamente adaptados precisam ser copiados para esta memória (através da função `cudaMemcpy`). Em seguida, o *kernel* é invocado, o que se refere à função de cálculo dos *grids* que é executada na GPU propriamente dita. Na implementação original, as dimensões x , y e z e espaçamento dos pontos do *grid* são definidas pelo usuário e a computação dos valores, tais como os da Equação 7, ocorrem em um laço triplamente aninhado percorrendo cada ponto das três dimensões. Na definição do *kernel*, esse laço é removido, pois durante sua execução N *threads* são despachados (N correspondendo, idealmente, ao número total de pontos do *grid*), cada um realizando, em paralelo e independentemente, os cálculos correspondentes a um ponto do *grid*. Ao fim do *kernel*, os *threads* são sincronizados, isso é, o dispositivo aguarda a finalização de cada um antes de devolver o fluxo de execução à CPU. Os resultados são copiados para a memória primária e a alocação na placa de vídeo é prontamente liberada, finalmente dando continuidade à execução do pyLiBELa.

O código resultante foi integrado ao programa pyLiBELA e está disponível publicamente através do repositório Github e também para uso através do Google Colab.

2.3 Aprendizagem de máquina

Para os fins do LiBELa, os *grids* sintetizam informações sobre energias de interação no sítio de interação em respeito apenas à molécula receptora. Porém, também pode-se calcular o *grid* do ligante, centrado no seu centro de massa, de forma a se obter duas representações de tamanho fixo e informações complementares sobre a energia de interação. Uma representação gráfica dos *grids* calculados para uma proteína e seu ligante está mostrada na Figura 2 abaixo.

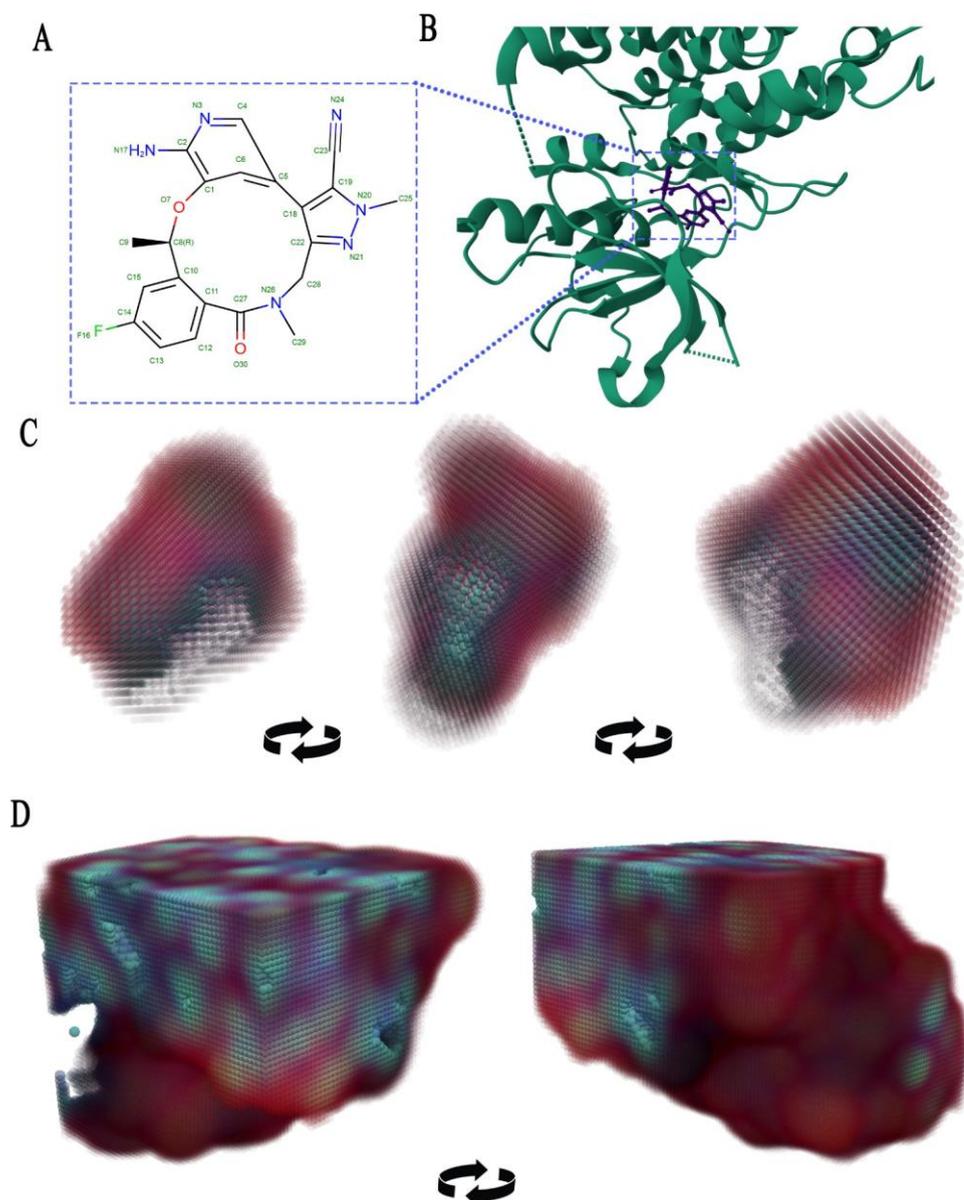


Figura 2 - Exemplo de representação, em *grids*, de uma proteína de quinase de linfoma anaplásico (ALK), código PDB 5AA9, e do seu ligante $C_{21}H_{19}FN_6O_2$ (ID 5P8) [22]. **(A)** Representação planar do ligante 5P8. **(B)** Estrutura tridimensional do complexo proteína-ligante da ALK. **(C)** *Grid* do ligante 5P8, em três ângulos distintos. **(D)** *Grid* da proteína 5AA9, centrado no sítio de interação do complexo, em dois ângulos distintos. Nas imagens **(C)** e **(D)**, cada ponto do *grid* está representado, através dos canais RGB, a contribuição relativa de diferentes energias calculadas no respectivo ponto do *grid*. A intensidade do canal vermelho do ponto é proporcional à interação eletrostática (elec), a intensidade do canal verde é proporcional ao termo repulsivo de van der Waals (vdwA) e o canal azul ao termo atrativo de van der Waals (vdwB), conforme a Equação 7. Para fins de ilustração, os pontos dos *grids* muito distantes de algum átomo (onde apenas a interação eletrostática tende a dominar) foram ignorados, do contrário os *grids* são volumes completamente preenchidos.

Fonte: Compilado pelo autor.

Essa característica inerentemente imagética dos *grids* convida a utilização de um modelo de rede neural que seja propício para esse tipo de dado. Com isso, adotou-se um modelo inspirado na arquitetura AlexNet, uma rede neural convolutiva profunda originalmente introduzida na competição *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) demonstrando excelentes capacidades em problemas classificação de imagens 2D (23). Porém, por se tratarem de *grids* tridimensionais, foi necessário adaptar a arquitetura para utilizar as versões tridimensionais de convoluções e *Maxpool*.

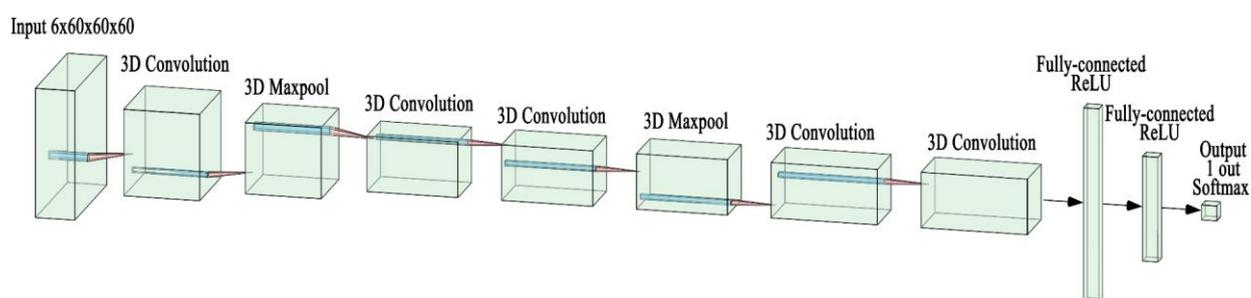


Figura 3 - Esquematisação da estrutura preliminar utilizada no modelo, inspirado na arquitetura AlexNet.

Fonte: Compilado pelo autor.

Diversas iterações do modelo com diferentes hiperparâmetros e camadas foram testadas, porém seguindo proximamente a estrutura básica demonstrada na Figura 3. O modelo elaborado consiste em uma série de camadas de convolução tridimensionais com um número crescente de filtros e decrescente de tamanho do *kernel*, interpostos com camadas de *Maxpool* tridimensional. Após essas etapas de convoluções, os dados de saída são linearizados a um vetor e submetidos a uma rede neural densa, com duas ou mais camadas profundas, com ativação ReLU, seguidas pela camada de saída, consistindo de um único escalar destinado à inferência da interação do complexo de entrada.

O conjunto de treinamento foi gerado a partir do PDBBind (versão 2020), um banco de dados que coleta resultados experimentais de ligantes biomoleculares, com informação estrutural disponibilizada no Protein Data Bank (PDB) (24). Esse conjunto possui 19,443 complexos proteínas-ligantes acompanhados de um valor de afinidade determinado experimentalmente. Desses valores, tomamos o oposto do logaritmo da constante de dissociação pela constante ou inibição ($-\log(K_d)$ ou $-\log(K_i)$) como alvos de treinamento do modelo. Utilizando a nova implementação em

CUDA, foram gerados os *grids* cúbicos com 30 Å de lado e com um espaçamento de 0,5 Å entre os pontos do *grid*, resultando em dimensões de 3x60x60x60 para cada um dos receptores e ligantes desses complexos. A primeira dimensão com tamanho 3 corresponde ao potencial eletrostático (elec), potencial atrativo de van der Waals (vdwA) e potencial repulsivo de van der Waals (vdwB), conforme a Equação 7.

Além disso, para cada complexo foram gerados 10 *grids* de *decoys* através de rotações e translações aleatórias do ligante, com *escore de ligação* ($-\log(K_d)$ ou $-\log(K_i)$) definido como 0. Esta estratégia de *data augmentation* resultou em um montante de mais de 2 TB de dados de treinamento. Os *decoys* são uma forma de expandir nosso conjunto de dados e introduzir uma restrição no treinamento do modelo, a fim de forçar a orientação correta do ligante como parte da inferência. O valor de entrada do modelo consiste na concatenação dos *grids* do receptor e ligante (ou *decoy*) em torno de seus três canais (elec, vdwA e vdwB), gerando um único *grid* de dimensões 6x60x60x60.

O modelo foi inicialmente implementado com a biblioteca Tensorflow, acompanhado de Keras, e posteriormente portado para PyTorch, ambos em Python. Os treinamentos dos modelos foram feitos com aceleração em GPU, primariamente em uma NVIDIA GeForce RTX 3060, porém também se fez uso de computação na nuvem por meio da plataforma do Google Cloud.

3 RESULTADOS

3.1 Performance em CUDA

A primeira etapa do projeto envolveu a implementação do cálculo dos *grids* de interação em linguagem CUDA. Uma vez realizada a implementação e a validação através de comparação dos *grids* calculados em CUDA com os *grids* calculados originalmente em CPU, passamos a avaliar a eficiência da implementação CUDA.

Para aferir o ganho de performance no processamento dos *grids* através da implementação de paralelismo em placa de vídeo, usando CUDA, fez-se um *benchmark* comparativo do tempo de execução necessário para calcular *grids* de diferentes dimensões, todos com espaçamento fixo dos pontos de 1 Å.

Tabela 1 - Média e desvio padrão do tempo de execução de grids de diferentes dimensões, com 100 replicações em cada, utilizando as funções de cálculo do grid implementadas com e sem aceleração em placa gráfica. Computação realizada em uma CPU AMD Ryzen 5 3600 e uma GPU NVIDIA GeForce RTX 3060.

Dimensões do <i>grid</i>	Tempo de execução em CPU (ms)	Tempo de execução em CUDA (ms)	<i>Speed-up</i>
10 x 10 x 10	65 ± 9	2,1 ± 0,9	31 x
20 x 20 x 20	530 ± 30	9 ± 3	59 x
30 x 30 x 30	1760 ± 60	13 ± 3	135 x
40 x 40 x 40	4200 ± 100	27 ± 5	156 x
50 x 50 x 50	8100 ± 100	46 ± 7	176 x

Fonte: Elaborada pelo autor.

Como demonstra a Tabela 1, notou-se pelos *benchmarks* um consistente aumento de performance ao utilizar a versão CUDA das funções de cálculo dos *grids*. Além disso, esse ganho de performance se torna mais expressivo à medida que se aumenta o número total de pontos do *grid*, que corresponde a um crescimento linear no tempo de execução em CPU, como evidenciado pelo Gráfico 1, atingindo melhorias de mais de 170 vezes na performance.

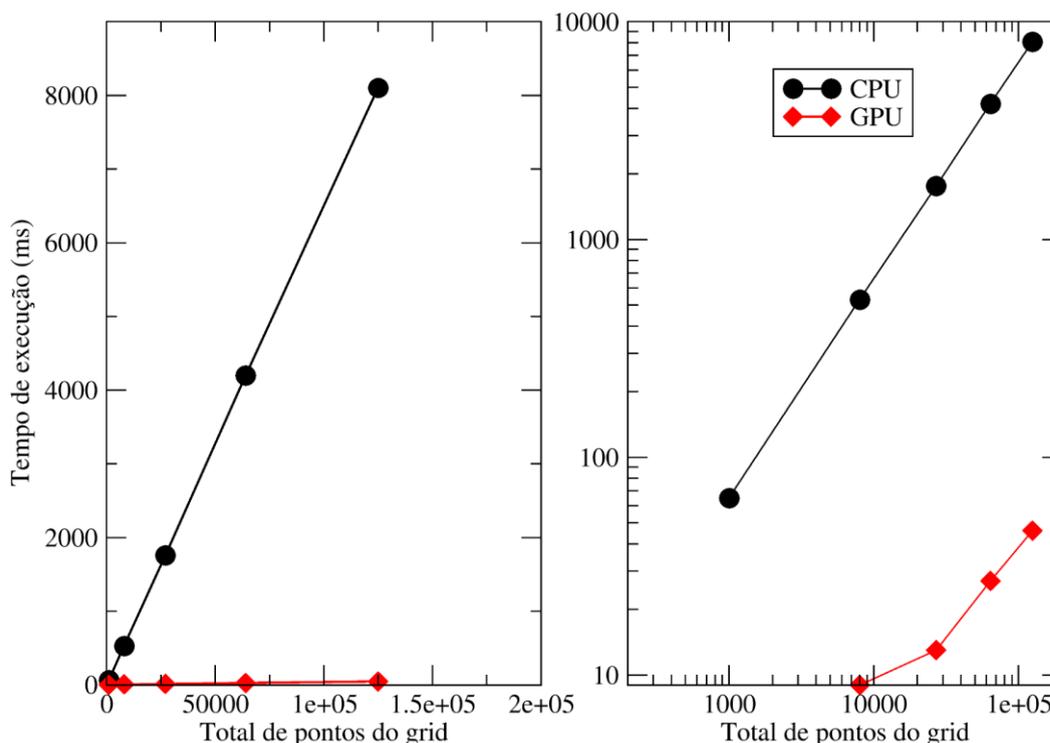


Gráfico 1 - Tempo de execução do *grid* com e sem processamento em GPU, em relação ao número total de pontos do *grid*. O gráfico à direita mostra os mesmos dados, porém em escala logarítmica.

Fonte: Elaborado pelo autor.

Esse resultado demonstra que softwares de *docking* molecular que utilizam representações de *grids*, onde o cálculo de um ponto seja independente do outro, podem instantaneamente se beneficiar de expressivos ganhos de performance ao optar pela execução em placas de vídeo.

3.2 Resultados do aprendizado de máquina

Os resultados obtidos na sessão anterior demonstraram que é possível calcular os *grids* contendo os potenciais eletrostático e de van der Waals para uma proteína como a lisozima do vírus T4, por exemplo, em menos de 10 segundos empregando a implementação CUDA. Com estes resultados animadores, passamos a desenvolver um modelo de aprendizado de máquina baseado em redes neurais convolutivas que pudessem aprender a afinidade de um ligante na forma de um escore de ligação dado por $-\log(K_d)$ a partir de *grids*. Inicialmente consideramos que a forma mais simples de iniciar seria empregar a representação da proteína na forma de um *grid* e o ligante na forma de um segundo *grid* e deixar a rede aprender que os *grids* devem ser complementares. Desde o início desta etapa do projeto entendemos que esta não era a forma ideal. O ideal seria obter uma rede que aprendesse a partir dos *grids* do receptor e das coordenadas atômicas

(e tipos atômicos) do ligante. No entanto, uma vez que este tipo de representação é um pouco mais complexa de se implementar, iniciamos com as representações de receptor e ligante em *grids* individuais.

3.2.1 Modelo CNN - TensorFlow

O conjunto de dados do PDDBind foi aleatoriamente sortido e separado em conjuntos de treinamento, teste e validação. Para etapas subsequentes de análise, foi selecionado um modelo do TensorFlow estruturado conforme a arquitetura da Figura 3, com convoluções de 64, 128, 128, 256 e 256 filtros, respectivamente, e duas camadas densas profundas de 128 unidades cada. Nele, utilizou-se um otimizador Adam, com taxa de aprendizado de 1.10^{-4} e erro quadrático médio (MSE) em relação ao escore de ligação experimental como função de perda. O modelo foi treinado por 6 épocas, em *batches* de 5 elementos, até atingir uma condição de parada antecipada onde o valor de perda parasse de crescer. Apesar de relativamente poucas épocas de treinamento, o decréscimo do valor de perda, demonstrado no Gráfico 2, indica que houve um grau de convergência no modelo.

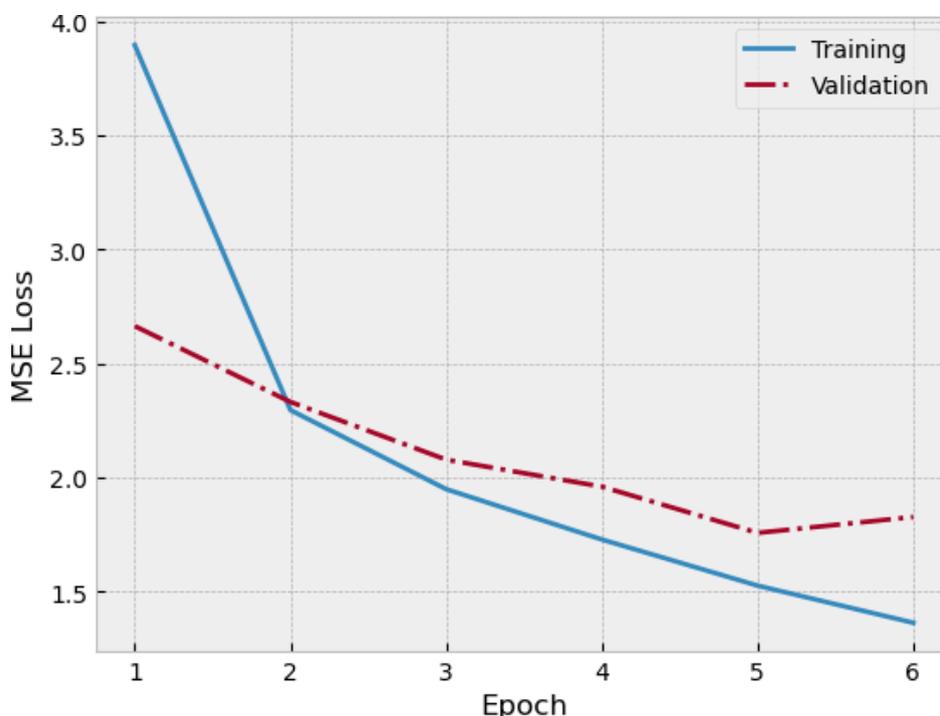


Gráfico 2 - Valor da função de perda, erro quadrático médio, ao longo do treinamento e validação do modelo.

Fonte: Elaborado pelo autor.

Para investigar mais profundamente a capacidade inferência do modelo obtido, gerou-se um conjunto de *grids* a partir de três proteínas (CXCR4, ADA e AA2AR) com ligantes e *decoys* selecionados do conjunto DUDE-Z (25). O conjunto DUDE-Z possui uma seleção de *decoys* desafiadores para uma série de complexos ligante-proteínas conhecidos, a fim de avaliar a performance de softwares de *docking* molecular (25). As conformações dos ligantes foram geradas usando o *docking* do pyLiBELa, a partir dos quais criaram-se os *grids* submetidos ao modelo finalizado para gerar previsões. O conjunto de inferências geradas pode ser organizado quanto ao valor, que representa uma quantia análoga ao $-\log(K_d)$, expressando quais ligantes possuem melhor afinidade. A partir desses resultados, podemos gerar uma classificação binária de bons ligantes (classe positiva) e *decoys* (classe negativa) se estabelecermos um valor de corte, dos quais quantias superiores representam bons ligantes e inferiores representam os *decoys*. Com esse procedimento, pode-se calcular a curva ROC associada ao desempenho do modelo para cada proteína

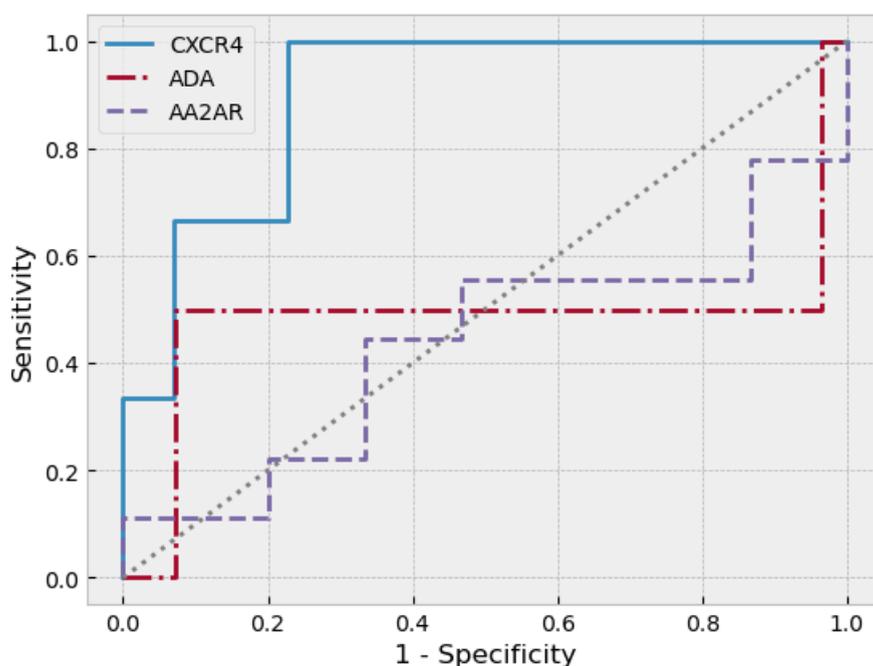


Gráfico 3 - Curva ROC (*Receiver Operating Characteristic*) associada às previsões das proteínas CXCR4, ADA e AA2AR.

Fonte: Elaborado pelo autor.

Os resultados mostram que a inferência realizada para um conjunto pequeno de moléculas (ligantes e *decoys*) resultou em um bom enriquecimento para a proteína CXCR4, mas em enriquecimentos fracos para as proteínas ADA e AA2AR. Em uma análise mais profunda, verificou-se que uma das razões para esse desempenho está atrelado ao fato dos *grids* dos ligantes serem gerados segundo o resultado do *docking* do LiBELa, que por vezes não alcança conformações ideais. Observamos que em diversos casos, o *docking* realizado antes da inferência com o modelo CNN chegou em poses incorretas para os ligantes, fazendo com que estes tivessem valores de energia próximo ao valores dos *decoys* e fazendo também com que não pudessem ser discriminados pelo modelo CNN. Como o modelo foi treinado de forma a ser totalmente sensível à posição do ligante, os *grids* gerados por *dockagens* incorretas criam um inerente fator limitante na capacidade de classificação. Essa limitação salienta uma prospectiva implementação a ser explorada no futuro do projeto, onde o ajuste conformacional seja uma parte integrante do aprendizado do modelo, gerando os *grids* necessários no próprio procedimento de inferência.

4 CONCLUSÕES E CONSIDERAÇÕES FINAIS

O *docking* molecular perdura como ferramenta essencial ao planejamento de fármacos. Existem uma diversidade de métodos e algoritmos que são amplamente utilizados nesses softwares, com distintas vantagens e desvantagens. Neste trabalho, foi demonstrado como a utilização de placas de vídeo, uma peça de *hardware* comum e comercialmente acessível, pode promover substanciais aumentos de performance em uma das técnicas utilizadas em *dockings* baseados em campos de força, com *grids* de potenciais pré-calculados, por meio da implementação de paralelismo, como foi aplicado ao pyLiBELa.

Tendo em mente as novas soluções proporcionadas pelo aprendizado de máquina à difícil tarefa de se determinar a interação de compostos com biomoléculas, avaliou-se a capacidade dos *grids* em codificar informações significativas da interação dos complexos, que pudessem ser resolvidos por uma rede neural. Construindo e treinando uma arquitetura CNN tridimensional, foi demonstrado a capacidade de um modelo convergir sob os parâmetros apresentados, evidenciado pela queda do valor de perda, a partir de um conjunto de dados contendo complexos ligante-proteínas enriquecidos com *decoys*.

A limitação do modelo em depender de conformações bem posicionadas de ligantes prévio à criação dos *grids* de entrada convida o aprimoramento da arquitetura a fim de conseguir inferir conformações ideais como parte operante do seu treinamento. Essa proposição, aliado a inovações nas formas de se criar representações treináveis no espaço molecular, são futuras perspectivas ao contínuo desenvolvimento do projeto.

REFERÊNCIAS

- 1 ANWAR, T.; KUMAR, P.; KHAN, A. Modern tools and techniques in computer-aided drug design. *In*: COUMAR, M. S. (ed.). **Molecular docking for computer-aided drug design: fundamentals, techniques, resources and applications**. Amsterdam: Elsevier, 2021. p. 1–30.
- 2 YU, W.; MACKERELL, A. D. Computer-aided drug design methods. **Methods in Molecular Biology**, v. 1520, p. 85–106, Nov. 2016.
- 3 MENG, X.-Y. *et al.* Molecular docking: a powerful approach for structure-based drug discovery. **Current Compute Aided-Drug Design**, v. 7, n. 2, p. 146–157, June 2011.
- 4 MUNIZ, H. S.; NASCIMENTO, A. S. Ligand- and receptor-based docking with LiBELa. **Journal of Computer-Aided Molecular Design**, v. 29, n. 8, p. 713–723, July. 2015.
- 5 ANTÔNIO, L.; NASCIMENTO, A. S. MolShaCS: a free and open source tool for ligand similarity identification based on Gaussian descriptors. **European Journal of Medicinal Chemistry**, v. 59, p. 296–303, Jan. 2013.
- 6 O'BOYLE, N. M. *et al.* Open Babel: an open chemical toolbox. **Journal of Cheminformatics**, v. 3, n. 1, Oct. 2011.
- 7 WEINER, P. K.; KOLLMAN, P. A. AMBER: Assisted model building with energy refinement: a general program for modeling molecules and their interactions. **Journal of Computational Chemistry**, v. 2, n. 3, p. 287–303, 1981.
- 8 VERKHIVKER, G. M. *et al.* Towards understanding the mechanisms of molecular recognition by computer simulations of ligand-protein interactions. **JMR. Journal of Molecular Recognition**, v. 12, n. 6, p. 371–389, Nov. 1999.
- 9 MENG, E. C.; SHOICHET, B. K.; KUNTZ, I. D. Automated docking with grid-based energy evaluation. **Journal of Computational Chemistry**, v. 13, n. 4, p. 505–524, May 1992.
- 10 SOUZA, J. V.; NOGUEIRA; V. H. R.; NASCIMENTO, A. S. Ligand binding free energy evaluation by Monte Carlo Recursion. **Computational Biology and Chemistry**, v. 103, p. 107830–107830, Apr. 2023.
- 11 LUTY, B. A. *et al.* A molecular mechanics/grid method for evaluation of ligand–receptor interactions. **Journal of computational chemistry**, v. 16, n. 4, p. 454–464, Apr. 1995.
- 12 JUMPER, J. *et al.* Highly accurate protein structure prediction with alphafold. **Nature**, v. 596, n. 7873, p. 583–589, July 2021.
- 13 BAEK, M. *et al.* Accurate prediction of protein structures and interactions using a three-track neural network. **Science**, v. 373, n. 6557, p. 871–876, Aug. 2021.
- 14 ABRAMSON, J. *et al.* Accurate structure prediction of biomolecular interactions with AlphaFold 3. **Nature**, v.630, p. 493–500, May 2024.

- 15 CHING, T. *et al.* Opportunities and obstacles for deep learning in biology and medicine. **Journal of the Royal Society Interface**, v. 15, n. 141, p. 20170387, Apr. 2018.
- 16 ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: FLEET, D. *et al.* (ed.) **Computer vision – ECCV 2014**. Cham: Springer, 2014. p. 818–833. (Lecture notes in computer science, v. 8689)
- 17 KORANNE, S. Boost C++ libraries. In: **Handbook of open-source tools**, Boston: Springer, 2011. p. 127–143.
- 18 VAN DER WALT, S.; COLBERT, S. C.; VAROQUAUX, G. The NumPy array: a structure for efficient numerical computation. **Computing in Science & Engineering**, v. 13, n. 2, p. 22–30, Mar. 2011.
- LAM, S. K.; PITROU, A.; SEIBERT, S. Numba. **Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC - LLVM '15**, 2015.
- 20 SANDERS, J.; KANDROT, E. **CUDA by example: an introduction to general-purpose GPU programming**. Sydney: Pearson Education, 2010.
- 21 GARLAND, M. *et al.* Parallel computing experiences with CUDA. **IEEE Micro**, v. 28, n. 4, p. 13–27, July 2008.
- 22 SHAW, A. T. *et al.* Resensitization to Crizotinib by the Lorlatinib ALK resistance mutation L1198F. **New England Journal of Medicine**, v. 374, n. 1, p. 54–61, Jan. 2016.
- 23 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. ImageNet classification with deep convolutional neural networks. **Communications of the ACM**, v. 60, n. 6, p. 84–90, May 2012.
- 24 LIU, Z. *et al.* PDB-wide collection of binding data: current status of the PDBbind database. **Bioinformatics**, v. 31, n. 3, p. 405–412, Oct. 2014.
- 25 STEIN, R. M. *et al.* Property-unmatched decoys in docking benchmarks. **Journal of Chemical Information and Modeling**, v. 61, n. 2, p. 699–714, Jan. 2021.