BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF PHYSICS

SPECIALIZATION PHYSICS WITH COMPUTER SCIENCE

BACHELOR'S THESIS

Supervisor

Lect. dr. E. Vințeler

Graduate

V.I. Macovei

2023

BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA

FACULTY OF PHYSICS

SPECIALIZATION PHYSICS WITH COMPUTER SCIENCE

BACHELOR'S THESIS

EXPLORING PHYSICS FRONTIERS THROUGH AI-DRIVEN DISCOVERY

Supervisor

Lect. dr. E. Vințeler

Graduate

V.I. Macovei

2023

Abstract

The escalating time and financial demands characteristic of contemporary research in Science, Technology, Engineering, and Mathematics (STEM) fields are largely attributable to the opportunistic nature of research and the sheer volume of requisite experiments. Artificial intelligence (AI), particularly machine learning, is posited as a potential tool to mitigate these challenges. The research and development process can be broadly classified into three stages: discovery, simulation, and experimentation. While human involvement is intrinsic to the experimentation phase and the simulation stage has already reached a significant level of advancement through high-performance computing, the discovery phase offers a window for substantial improvements through AI. The thesis at hand aims to explore and validate this proposition by examining potential techniques and gauging the feasibility of this idea in scenarios where considerable resources are not readily accessible.

The first use case involves the Varikon Box, a puzzle sharing some similarities yet distinct differences with the Rubik's Cube. This case study utilizes deep reinforcement learning, a first-of-its-kind application to the Varikon Box. This puzzle presents a microcosmic analogy to numerous problems in STEM, specifically permutation problems with limited feedback, and offers unique combinatorial challenges. Despite not being optimal in terms of the number of moves, the algorithm demonstrated close to 100% success rate in solving the puzzle.

The second part of this investigation revolves around IBM's MOLFORMER-XL project, a remarkable breakthrough in molecule generation and molecular property prediction. IBM provides a pre-trained model, trained on the PubChem and ZINC datasets. All previous IBM experiments using MOLFORMER-XL have employed a configuration of 16 V100 GPUs, an expensive and hard-to-come-by setup in most research centers. A key question this thesis addresses is the feasibility of fine-tuning this model using a single V100 GPU to reduce infrastructure costs. This investigation concludes positively, demonstrating that fine-tuning models can predict molecular properties effectively, even when operating on a single GPU. These fine-tuned models can also be used to create vector representations of molecules for tasks such as chemical space visualization, achieving GPU utilization over 75%. This was accomplished by fine-tuning the pre-trained model provided by IBM on the classification and regression benchmarks from MoleculeNet, yielding a final area under the receiver operating characteristic curve of 0.85. These results underscore the efficacy and feasibility of AI application in STEM research, even in low-resource settings.

Contents

Abstract **2**

Introduction **4**

1 Theoretical Foundations Part I **8**

1.1 Knowledge Discovery in Databases (KDD) 8

1.2 Introduction to Machine Learning (ML) 10

1.3 Deep Reinforcement Learning (DRL) 12

2 Use Case 1: DRL for the Varikon Box **15**

2.1 The Varikon Box and its Unique Characteristics 15

2.2 Representing the Varikon Box 16

2.3 Implementing Moves 17

2.4 Solving the Varikon Box 17

2.5 Implementation 18

3 Theoretical Foundations Part II **21**

3.1 Natural Language Processing (NLP) 21

3.2 Large Language Models (LLMs) 24

3.3 Chemoinformatics 26

4 Use Case 2: MoLFormer-XL **28**

4.1 Project Overview 28

4.2 Experiments 29

5 Technologies **40**

Conclusions **43**

A List of Acronyms **44**

Bibliography **45**

Introduction

Background

In today's digital age, natural language documents stand as a vast yet unstructured data source that poses significant challenges to machine systems' comprehension. The constantly burgeoning volume of such textual content on the internet necessitates automated text analysis, especially within Research and Development (RD) operations. In the current technological landscape, Natural Language Processing (NLP), a subset of artificial intelligence (AI), is instrumental in making natural language-based information accessible to machine systems. With the latest advancements in this field, Large Language Models (LLMs) have found applications in predicting protein structures at an atomic level and generating novel molecular structures from basic representations.

Motivation

Current physics research methodologies predominantly hinge on human discovery, serving as the starting point for simulations executed on High Performance Computing (HPC) systems. The primary purpose of these simulations is to refine and validate ideas mathematically before they undergo experimental testing, aiming to eventually produce a novel product (for example, a new material). However, this process often demands a considerable amount of time, stretching over several years in some instances.

The motivation behind this thesis is to propose a new approach to accelerate product discovery. The envisioned approach aims to supplement human input with AI's capabilities, opening the door for exploring innovative possibilities. The intention is to dedicate experimental efforts to only the most promising ideas, enhancing the likelihood of their validation during the experimental phase, and thus accelerating the path towards the final product creation.

Research Questions and Objectives

This thesis intends to identify new methods for accelerating physics research through the application of AI. Furthermore, the potential application of Generative AI, particularly Large Language Models, in molecular physics will be explored using real data. The key objectives are:

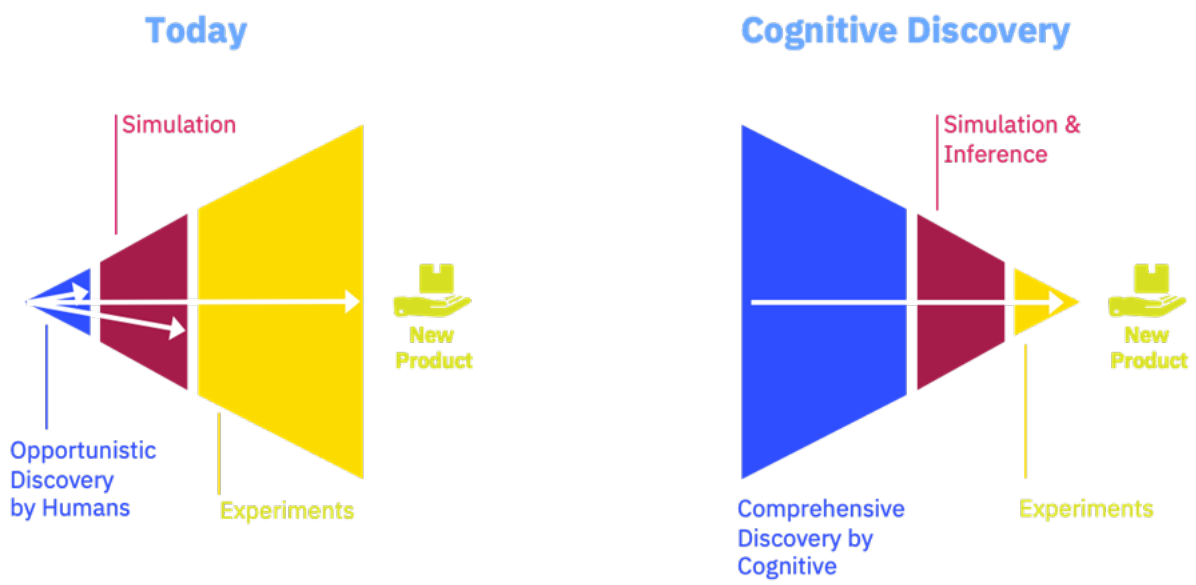

Fig. 1: A disruptive approach to research and development.

1. To ascertain the capabilities of AI in scrutinizing molecular structure aspects, elemental composition, and other critical parameters to predict properties like stability, bandgap, conductivity, and catalytic activity.

2. To probe the feasibility of conducting experiments with Recurrent Neural Networks (RNNs) and Transformers on limited infrastructure resources and evaluate the potential impact on the research process.

3. To explore the possibility of using pre-trained models for molecular property prediction, evaluating their strengths, weaknesses, and potential fine-tuning for improved performance.

4. To scrutinize the scalability of these models, particularly when managing large datasets, and assess the computational resources needed for their effective deployment.

Moreover, in the field of physics research using large datasets with Foundation Models will have direct implications on the following six areas:

Data generation and uugmentation: Physics research often relies on access to large and diverse datasets. Generative AI can generate synthetic data that mimics the characteristics of real-world physics data. This ability to generate data allows researchers to augment their datasets, especially in cases where obtaining sufficient real data is challenging or expensive. It enables them to explore a broader range of scenarios, validate hypotheses, and enhance the statistical significance of their findings.

Hypothesis testing and simulation: Generative AI can be utilized to simulate physical phenomena and test hypotheses. By training generative models on known physical principles, researchers can generate simulated data that exhibits the same behavior as real-world observations. This provides a powerful tool for hypothesis testing, exploring different scenarios, and gaining

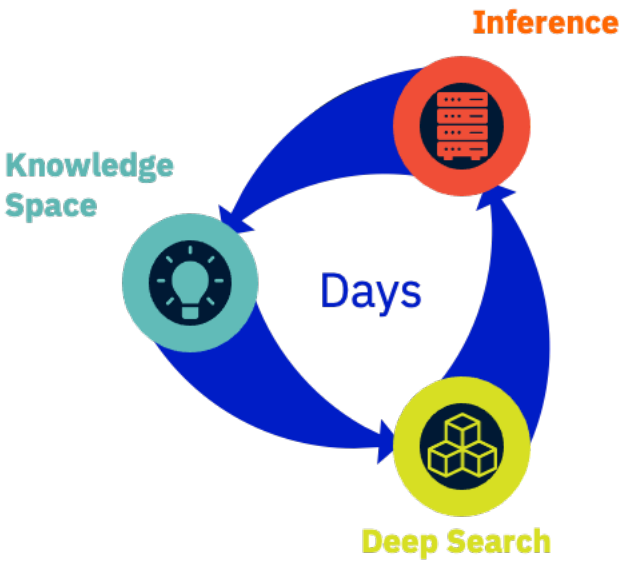

Fig. 2: Knowledge discovery pipeline.

insights into complex physical systems that may be difficult to study through analytical or experimental means alone.

Enhancing Experimental Design: Generative AI can assist in optimizing experimental designs. By generating synthetic data that covers a wide range of possible outcomes, researchers can analyze the potential performance of different experimental setups and identify optimal configurations. This aids in minimizing experimental costs and time by providing insights into the expected outcomes before conducting actual experiments.

Data reconstruction and denoising: Physics experiments often involve noisy or incomplete data. Generative AI models can be trained to reconstruct missing or noisy data and denoise experimental observations. By learning the underlying patterns and structures in the data, these models can fill in gaps or remove noise, enabling researchers to obtain more accurate and reliable information from their experimental measurements.

Novel discovery and design: Generative AI can assist in the discovery and design of novel physical systems, materials, or structures. By generating diverse variations of existing structures or by exploring latent spaces of generative models, researchers can identify new configurations or optimize existing ones. This can lead to the discovery of materials with desired properties, novel physical phenomena, or innovative designs that push the boundaries of current understanding.

Conceptual understanding and visualization: Foundation Models can aid in the visualization and conceptual understanding of complex physical systems. By generating visual representations of physical phenomena or by mapping latent spaces of generative models, researchers can gain insights into the underlying principles and relationships governing the system. This can facilitate the development of intuitive explanations, visualizations, and educational tools for communicating scientific concepts to a broader audience.

Course of Investigation

The investigation in the thesis at hand unfolds through four main chapters, followed by a concluding section. After laying down the groundwork in the introduction, each chapter delves deeper into specific aspects of the study.

Chapter 1: Theoretical Foundations Part I delves into the theoretical backbone of this study. It starts with an introduction to knowledge discovery in databases (KDD) and proceeds to a detailed exploration of machine learning (ML). The chapter culminates with a comprehensive overview of deep reinforcement learning (DRL), ensuring a strong theoretical foundation for the investigations to follow.

Chapter 2: Use Case 1: DRL for the Varikon Box shifts the focus from theory to a practical application of DRL, exploring the unique characteristics of the Varikon Box. This chapter describes the representation and implementation of moves, and ultimately how to solve the Varikon Box using DRL techniques, demonstrating the real-world applications of the previously discussed theoretical constructs.

Chapter 3: Theoretical Foundations Part II returns to theory, this time focusing on natural language processing (NLP), large language models (LLMs), and Chemoinformatics. The discussions in this chapter lay the groundwork for the application of these concepts in chemoinformatics, which plays a central role in the next practical use case.

Chapter 4: Use Case 2: MoLFormer-XL details the application of the concepts discussed in Chapter 3. This chapter provides an overview of the MoLFormer-XL project and then delves into the specifics of the experiments conducted. The application of LLMs and chemoinformatics concepts in a real-world scenario underscores the practical relevance of the theoretical discussions.

Chapter 5: Technological Aspects provides an overview of the pivotal technology that underpins the practical work presented in this thesis, specifically, the IBM AC922 System. It discusses the system's capabilities, particularly its AI potential, which makes it an optimal choice for complex computations involved in the fine-tuning process of the MOLFORMER-XL model. This short but essential chapter underlines the significance of appropriate technological infrastructure in achieving rigorous and accurate research outcomes.

The thesis concludes by summarizing the key findings from the study and providing a forward-looking perspective aligned with the objectives of the research and the wider research field.

1 Theoretical Foundations Part I

The primary focus of the first chapter is to lay down the theoretical underpinnings of artificial intelligence (AI), with a specific emphasis on NLP. Given the vast array of applications and the extensive versatility inherent to NLP, the current chapter concentrates on those aspects that are most pertinent to the subject matter of the thesis. These include but are not limited to KDD, the cross-industry standard process for data mining (CRISP-DM), various training types and the technologies and methodologies used. In light of recent advancements in NLP, neural networks are also introduced, with a focus on feed forward neural networks (FNN)s, recurrent neural networks (RNN)s, and transformers. For a more exhaustive exploration of the field, readers may refer to works such as [1], [2], [3] or [4].

1.1 Knowledge Discovery in Databases (KDD)

In the pursuit of new knowledge, one often finds themselves navigating vast oceans of data. The challenge lies not just in the sheer volume of said data, but also in the complexity and diversity of the information it contains. That is the reason why the concepts of KDD and data mining (DM) play a pivotal role in scientific discovery. KDD refers to *the nontrivial extraction of implicit, previously unknown and potentially useful information from data [...]* [5]. It's a comprehensive process that involves interpreting complex datasets to discover meaningful patterns and relationships. On the other hand, DM is a subset of the KDD process. It involves the application of various techniques, such as machine learning and statistics, to identify and extract these patterns from the data. While KDD provides the overarching framework, DM offers the practical tools for the extraction of knowledge.

Having established the importance of KDD and DM, one can discuss the specific methodology of the KDD process: the CRISP-DM. An iterative procedure, the CRISP-DM comprises six

primary stages which can be visualised in Fig. 1.1 and are expanded on below.

1. **Business Understanding:** The initial stage involves comprehending and analyzing the project from a business perspective with the aim of translating the overall problem into a data mining problem.
2. **Data Understanding:** The second stage starts with data collection, which provides preliminary insights into the data, enables the formulation of initial hypotheses, and helps identify potential issues. It also allows for the option to *redefine the whole problem and return to the business understanding* stage if necessary [6].
3. **Data Preparation:** Various techniques are used to obtain a final dataset that can be forwarded to the next phase. It often involves dividing the data into subsets for training, testing, and potentially validation.
4. **Modeling:** The fifth stage involves applying various models to the prepared data. Some models may require specific input formatting, necessitating a return to the data preparation stage.
5. **Evaluation:** Before deploying the model, it is crucial to evaluate its performance using metrics that align with the business goals.
6. **Deployment:** In the final stage, the model is integrated into production. However, for the sake of brevity, the deployment phase will not be further discussed.

Fig. 1.1: The six stages of a CRISP-DM project [6].

The primary focus of the current thesis is on the modeling and evaluation stages, by utilising the MOLFORMER-XL model, a product of machine learning, and seeking to enhance it using reinforcement learning techniques. The established objectives require a proper understanding of the AI techniques, reason why the following section provides an in-depth overview of the field.

1.2 Introduction to Machine Learning (ML)

In the realm of science and technology, the term *artificial intelligence* has become a common part of the modern lexicon. Yet, its relationship with ML is often less understood. To clarify, *artificial intelligence is the broader concept of machines being able to carry out tasks in a way that it would be considered smart [...]* [7]. ML, on the other hand, is a specific application of AI. It revolves around the idea that machines should not just perform tasks, but also learn from the data they process. This concept of experience is central to the definition of ML, as articulated by Tom Mitchell: *a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E* [8].

The utility of AI and ML is vast and varied, spanning numerous fields and industries. From predicting weather patterns to diagnosing diseases, these technologies have revolutionized the way we process information and make decisions. Their specific applications, however, depend on the type of learning they employ, leading to the classification of ML into four major types, discussed in the following sections.

1.2.1 Supervised Learning

The moniker supervised learning is derived from the role of data scientists as guides, instructing the algorithm on the desired outcomes. It necessitates labeled data, complete with the correct answers that the algorithm is expected to produce. Supervised learning is further divided into several categories which are not mutually exclusive, as a single task might be framed as multiple types of problems.

- **Classification** involves predicting a discrete label. For instance, determining whether an email is spam or not is a binary classification problem, while determining what type of fruit is shown in a picture is a multi-class classification problem
- **Regression** involves predicting a continuous value (e.g. predicting a person's age or the temperature the next day)
- **Ranking** is a type of supervised learning where the output is a list of items in some type of order. (e.g. a search engine uses ranking algorithms to determine the order in which search results are displayed)
- **Sequence prediction** has the main objective of predicting the next values in a sequence based on the previous ones. It is commonly used in language modeling (predicting the next word in a sentence) and time series forecasting.

1.2.2 Unsupervised Learning

Unsupervised learning represents a form of ML that closely aligns with the concept of *true AI*. It involves machines learning to identify complex patterns and processes without explicit guidance or supervision. This approach is particularly beneficial when the data does not include predefined targets or when the nature of the patterns to be discovered is not known in advance. One common application of unsupervised learning is **clustering**, where the goal is to group similar items together based on their inherent characteristics. For instance, news articles could be clustered based on their content or genre, allowing for more efficient information retrieval and organization.

Despite the inherent complexity and implementation challenges associated with this form of ML, it has the potential to address problems that are typically beyond human reach. However, its widespread application has been somewhat limited compared to supervised learning. Recent advancements in the field of LLMs, particularly the development of transformer architectures, have shown promising results, suggesting new possibilities for collaborative research between humans and machines.

1.2.3 Semi-Supervised Learning

In many practical cases, the process of labeling can be prohibitively expensive, and for large datasets, it can become a tedious and time-consuming endeavor. Moreover, providing an abundance of labeled data might unintentionally instill human prejudices into the model. That is where semi-supervised machine learning finds its niche. It serves as a middle ground between supervised and unsupervised learning, leveraging the strengths of both. Semi-supervised learning operates on a mix of labeled and unlabeled data, typically a small amount of the former and a larger amount of the latter. The idea is to use the labeled data to guide the learning process and the unlabeled data to provide a broader context, thereby enhancing the model's ability to generalize from limited examples. This approach not only helps to conserve resources but also mitigates the risk of overfitting to the labeled data.

1.2.4 Reinforcement Learning (reinforcement learning (RL))

RL holds its place as one of the key pillars of ML, along with supervised and unsupervised learning. At the core of RL lies the premise of an autonomous agent striving to optimize its behavior in a given environment. This environment is defined by a set of possible states and actions, with the aim of maximizing rewards and minimizing any form of risk. The framework of RL comprises five fundamental elements: the **agent** (the autonomous entity that learns and performs actions), the **environment** (the context or domain where the agent operates), multiple **states** (situations or condition of the agent within the environment at certain times), various **actions** (decisions taken by the

agent, influencing the current state of the environment) and the **reward** (the feedback mechanism, providing positive or negative reinforcement based on the agent's actions). The learning process in RL typically follows a sequence of four key iterable steps:

1. The agent maintains observation of the environment's input state.
2. Decisions are made, and actions are taken based on the agent's decision-making function.
3. After an action is performed, the agent receives a reward.
4. Information regarding the reward state is collected and stored for future reference.

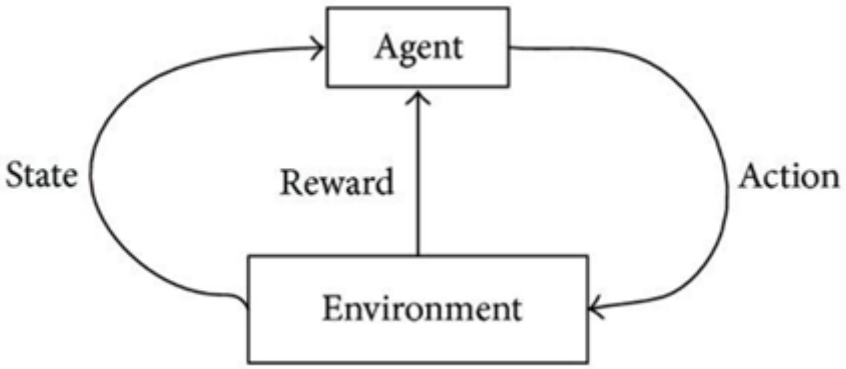

Fig. 1.2: Reinforcement Learning Components [9]

Reinforcement Learning tasks are generally categorized into two types. **Episodic tasks** have a distinct start and end point, leading to a series of events, or episodes, consisting of states, actions, and rewards. In contrast, **continuous tasks** tasks do not have a terminal state and continue indefinitely. The agent concurrently interacts with the environment and learns to choose the best actions.

RL algorithms have wide-ranging applications, with some of the more notable ones being the Monte Carlo methods and temporal difference learning. Monte Carlo methods gather rewards at the conclusion of each episode before computing the maximum expected future reward, while temporal difference Learning estimates rewards step by step. In the present thesis, RL and Monte Carlo tree search (MCTS) are employed to solve a specific permutational problem concerning the Varikon Box, which presents a unique challenge due to its scarce reward structure.

1.3 Deep Reinforcement Learning (DRL)

DRL is an advanced branch of AI that synergistically combines deep learning and RL. DRL differentiates itself from traditional RL by using deep neural networks (DNNs) for feature extraction

and function approximation, enabling it to handle high-dimensional state spaces and learn directly from unprocessed sensory data. This feature is particularly beneficial in complex environments. However, a deeper understanding of deep learning, which will be explored in the following section, is crucial to fully appreciate the workings and potential of DRL.

1.3.1 Deep Learning

Deep learning is defined as the subset of ML that employs artificial neural networks (ANNs) with a minimum of three layers to model and discover intricate trends within datasets. ANNs form the backbone of deep learning and are inspired by the biological neural circuit. Each network consists of interconnected layers of nodes, or *neurons*, each of which *is responsible for interpreting a specific pattern within the data* [10].

In DRL, neural networks are used to approximate the value functions or policies, enabling the system to make decisions that maximize the cumulative reward. The ability of neural networks to learn and generalize from complex, high-dimensional data makes them an indispensable component of DRL. In essence, deep learning, with its neural networks, provides the necessary tools for transforming and interpreting raw data, setting the stage for the reinforcement learning component to learn an optimal policy. This symbiotic relationship between deep learning and reinforcement learning is what empowers DRL to tackle complex, real-world problems.

1.3.2 Policy Gradient Methods

In DRL, policy gradient methods hold significant importance. While value-based methods attempt to learn the decision-making function (or the policy) indirectly, policy gradient methods optimize it directly. This direct approach is particularly beneficial in continuous action spaces, where finding the action that maximizes the value function can be computationally challenging.

Policy gradient methods operate on the principle of determining the gradient, which is the measure of change, of the expected total reward with respect to the policy parameters. Following this, the parameters are modified to coincide with the direction of the identified gradient, a process usually executed through gradient ascent. The gradient serves as an indicator of how a minor adjustment in the policy parameters will influence the expected total reward. By making alterations to the parameters in the direction of the gradient, the algorithm progressively refines the policy in an iterative manner.

1.3.3 Deep Q-Learning (DQN)

Q-learning is a value-based method in RL that seeks to learn a Q-function, which estimates

the cumulative reward for each action in each state, leading to the optimal policy. deep Q-learning (DQN) is an extension of Q-learning that uses a deep neural network to approximate the Q-function. Traditional Q-learning struggles with high-dimensional state spaces due to the so-called *curse of dimensionality* [11]. DQN addresses this issue by using the powerful function approximation and generalization capabilities of DNNs.

DQN has been successfully applied to a variety of tasks, demonstrating the power of combining deep learning with reinforcement learning. In the application of solving the Varikon Box DQN is particularly useful. The state space of the Varikon Box is extremely large, but can be handled by a DQN which learns to estimate the value of different moves from different configurations, guiding the agent towards solutions. By learning directly from raw configurations of the cube, a DQN can develop a policy for solving the Varikon Box without the need for hand-crafted features or explicit rules.

2

Use Case 1: DRL for the Varikon Box

This chapter concerns the solution for the 3x3x3 Varikon Box, employing the method of deep reinforcement learning (DRL).

2.1 The Varikon Box and its Unique Characteristics

The Varikon Box 3x3x3 puzzle shares a visual similarity with typical 3x3x3 sliding cube puzzles. It consists of a transparent box, housing 26 cubicles and an empty space, permitting the cubicles to move. The distinguishing characteristic of this puzzle lies in the fixed positioning of the face center cubicles onto the central Varikon Box cubicle, resulting in only 19 movable cubes, organized around the box's edges and corners.

These movable cubes are categorized into two types: corner cubicles, located at the box's corners, and edge cubicles, positioned between the corner cubicles on the box's edges. The faces of the cubicles are referred to as 'squares'. Each cubicle has three white squares and three blue squares, intersecting at opposing corners of the cubicle. The puzzle is in a solved state when all the outward-facing squares are blue, with the exception of the empty cubicle squares, which are classified as 'empty squares'. A single move consists of tilting the Varikon Box, resulting in the movement of two cubicles and placing the empty cubicle in a corner position in all possible states.

The fixed cubicles at the center of the Varikon Box faces are omitted from this discussion, focusing only on the moving cubicles. Therefore, the solved state manifests 45 blue outward-facing squares and 3 empty squares. Another requirement for the solved state is that the inward-facing squares adjacent to the empty cubicle must be white. The puzzle permits only even permutations, leading to a total of $8 \cdot 19!/2 = 486{,}580{,}401{,}635{,}328{,}000$ possible states, considering the constraint that the empty cubicle can only occupy corner positions.

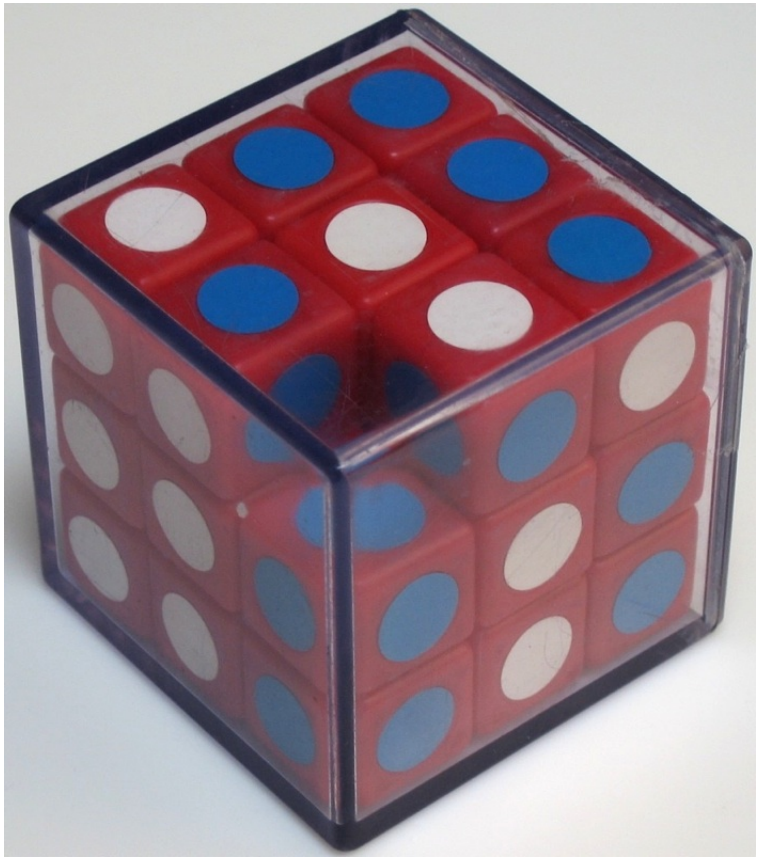

Fig. 2.1: The 3x3x3 Varikon Box, patented by D. Kosarek, 1974 [12].

Despite the non-empty cubicles being identical, their differing orientations make them distinct within the puzzle's representation. This ensures that an unsolved state has only one possible solved state, given the set of available cubicles. The empty cubicle assumes the position of the missing cubicle in the solved state, leading to a unique solved state for a specific Varikon Box orientation.

2.2 Representing the Varikon Box

The representation of the Varikon Box is central to this project, heavily reliant on orientation. The notations 'F' (front), 'L' (left), 'B' (back), 'R' (right), 'U' (up), and 'D' (down) represent the six faces of the Varikon Box, the six squares of each cubicle, and each of the potential moves. It should be noted that in this representation, the 'F' face is furthest from the viewer, while the 'B' face is closest. The term 'M' (middle) is occasionally used in the context of edge cubicles.

The Varikon Box is represented as a tensor of tensors, wherein the smaller 'cubicle tensors' represent individual cubicles. Cubicle tensors are indexed to denote their positioning. A total of 20 cubicle tensors are required, excluding the cubicles at the center of the Varikon Box faces and the central cubicle. However, the empty cubicle is considered, and its position is identified. White is represented by '0', blue by '1', and empty by '2'.

2.3 Implementing Moves

The Varikon Box puzzle allows for six potential moves: 'F' (front), 'L' (left), 'B' (back), 'R' (right), 'U' (up), and 'D' (down). At any given time, only three moves are valid. Each move's notation signifies the direction in which the empty cubicle moves.

When a move is executed, it triggers a circular shift of the cubicles in a 3x3 square around the empty cubicle's position. The direction of the shift depends on the move made. For instance, an 'F' move causes a clockwise shift of the cubicles in the front 3x3 square (when viewed from the front), while a 'B' move causes an anticlockwise shift of the cubicles in the back 3x3 square (when viewed from the back).

In addition to this shift in positioning, a move also affects the orientation of the edge and corner cubicles involved in the shift. Corner cubicles rotate in a fixed direction (clockwise or anticlockwise) depending on the move, while edge cubicles flip around their long axis. For each of the six potential moves, the implementing function will:

1. Identify the relevant 3x3 square of cubicles.
2. Calculate the new positions and orientations of the cubicles after the move.
3. Update the Varikon Box tensor to reflect these changes.
4. Record the move in the history of moves for that Varikon Box state.

2.4 Solving the Varikon Box

Solving the Varikon Box involves finding a sequence of moves that transforms the current state of the puzzle into the solved state. This can be a challenging task due to the large number of possible states. Therefore, a systematic approach is required.

A common approach for solving puzzles like this is to use a breadth-first search (BFS) algorithm. Starting from the current state, this algorithm explores all possible moves to generate new states, then repeats this process for each new state until it finds a state that matches the solved state. To speed up the search, the algorithm can use heuristics, which are strategies based on insight into the problem, to prioritize moves that are likely to lead to the solution faster. For the Varikon Box, potential heuristics could consist of prioritizing moves that increase the number of blue outward-facing squares or prioritizing moves that place the empty cubicle in a corner.

The BFS algorithm, combined with these heuristics, should be able to find a solution to the Varikon Box puzzle. However, due to the complexity of the puzzle, the solution may not be the

shortest possible sequence of moves. Optimizing the solution to find the shortest sequence would be a more challenging task that would likely require a more advanced algorithm.

2.5 Implementation

2.5.1 Overview

The implementation of the deep reinforcement learning algorithm for solving the Varikon Box is based on the adaptation of a similar algorithm used for solving the Rubik's Cube. The algorithm utilizes a neural network architecture and a batched weighted A* search (BWAS) algorithm to find the optimal solution.

2.5.2 Neural Network Architecture

The neural network architecture is defined in the network.py file. The architecture is composed of a CubeNet class, which is a subclass of the nn.Module class from the PyTorch library. The CubeNet class defines the structure of the neural network, which includes a sequence of linear layers, batch normalization layers, and ReLU activation functions. The network also includes a series of residual blocks, which are defined in the ResidualBlock class.

The forward method of the CubeNet class defines how an input tensor is processed through the network. The input tensor, which represents the state of the Varikon Box, is passed through the initial block of layers, then through each residual block, and finally through the final linear layer. The output of the network is a single value that represents the estimated cost to solve the Varikon Box from the given state.

2.5.3 Training the Network

The training process of the neural network is defined in the train.py file. The training process involves generating a batch of scrambled states of the Varikon Box, creating the corresponding target values, and updating the network parameters to minimize the difference between the network's output and the target values.

The train.py file also includes code for setting up the training environment, loading and saving the network parameters, and logging the training progress. The training process utilizes the Adam optimization algorithm and a learning rate scheduler from the PyTorch library.

Loss Graph: An exponential decay in the loss graph is a good sign as it indicates that the model is learning and improving its predictions over time. However, the slight increase in loss towards

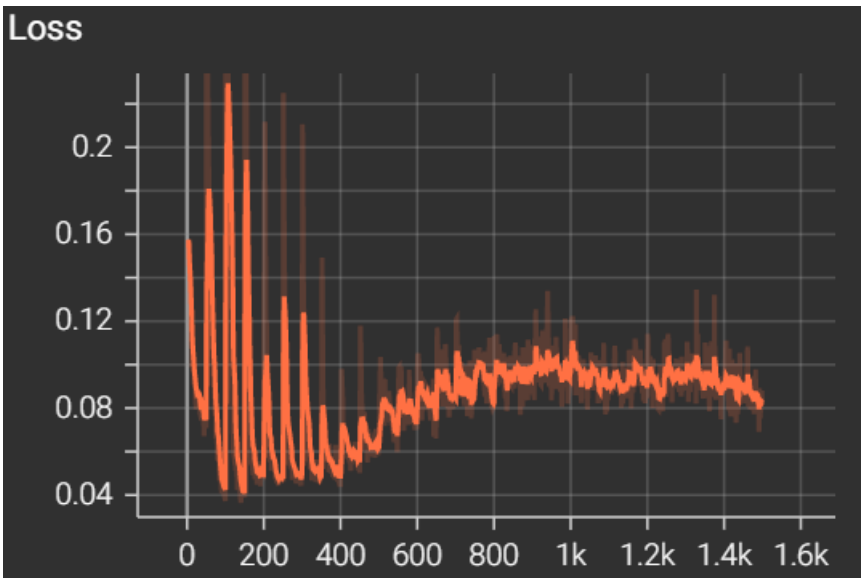

Fig. 2.2: Loss function.

the end could indeed be a sign of overfitting. Overfitting occurs when the model starts to memorize the training data rather than learning to generalize from it. This can lead to an increase in loss when the model is evaluated on new, unseen data. It's also possible that the model is experiencing some instability in its learning process towards the end of training.

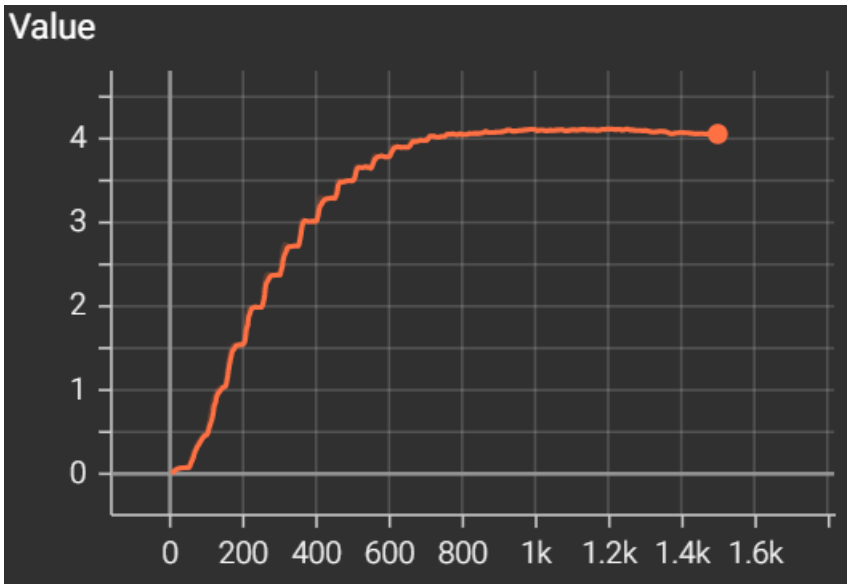

Fig. 2.3: value function.

Value Graph: A logarithmic increase in the value graph suggests that the model is gradually improving its performance, but the rate of improvement is slowing down over time. This is expected behavior as the model starts to converge to a solution. The fact that it appears to converge to a value suggests that the model has found a stable solution that doesn't change significantly with further training.

Time Graph: If the time graph looks like an isosceles triangle with the peak at epoch 700, it suggests that the time taken for each epoch initially increases, reaches a maximum at epoch 700,

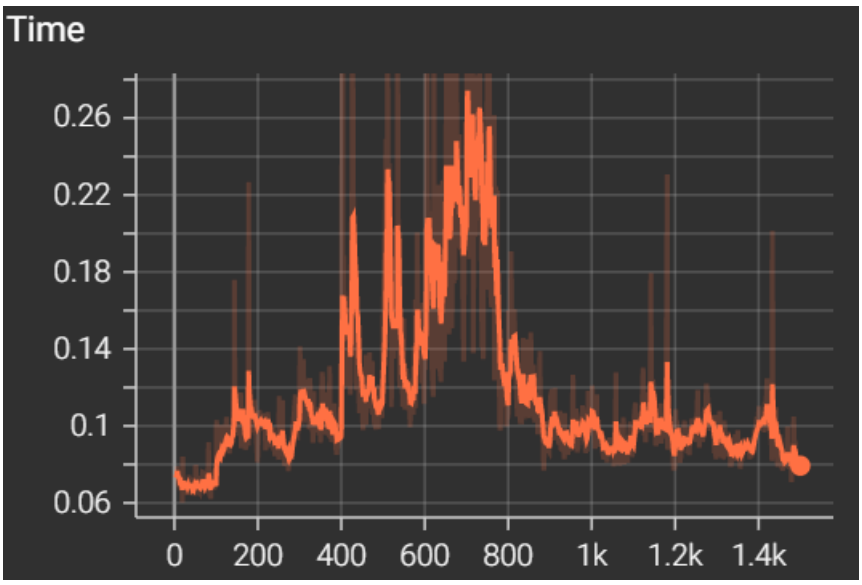

Fig. 2.4: Time function.

and then decreases.

2.5.4 Batched Weighted A* Search

The BWAS algorithm is implemented in the BWAS.py file. The algorithm starts with a scrambled state of the Varikon Box and explores the next possible states by applying all possible moves. The next states and their corresponding costs are estimated by the neural network and added to a priority queue. The algorithm then selects the state with the lowest cost from the queue and repeats the process until it finds a solved state.

The Node class, defined in the node.py file, represents a state in the search space. Each node includes information about the state, the move that led to the state, the depth of the node in the search tree, and whether the state is a solved state.

2.5.5 Testing the Network

The performance of the trained network is evaluated by testing its ability to solve scrambled states of the Varikon Box. The testing process involves generating a batch of scrambled states, applying the BWAS algorithm to find the solution for each state, and comparing the solutions with the optimal solutions. The testing process is also implemented in the train.py file. Further testing needs to be executed on the model, but in general it was able to consistently solve the Varikon Box in about 90 moves. The code can be found on GitHub at https://github.com/Vict0r-M/VarikonBox.git.

3

Theoretical Foundations Part II

3.1 Natural Language Processing (NLP)

Building upon the prior exploration of AI, ML, and DNNs, it is pivotal to understand the specific role of NLP within the technological panorama they provide. While AI, ML, and DNNs provide a solid foundation, these alone cannot thoroughly interpret and analyze human language in its complexity, diversity, and nuance. In this juncture NLP becomes crucial.

Fig. 3.1: From traditional AI to foundation models

NLP is a specialized discipline of AI that focuses on the interaction between computers and humans through language. Its goal is to read, decipher, understand, and make sense of the human language in a valuable way. This is accomplished through multiple methods, such as syntactic analysis, semantic analysis, discourse analysis, and pragmatic analysis. Syntactic analysis investigates the structural patterns of language, while semantic analysis seeks to understand meaning. Discourse analysis looks at language in context and pragmatic analysis considers both linguistic context and shared world knowledge. These methods collectively enable NLP to interpret human language in a comprehensive manner. In a broader perspective, NLP is a bridge between machines

and humans, enabling a more effective and efficient interaction, which is crucial in a myriad of applications, including but not limited to speech recognition, language translation, sentiment analysis, and information extraction, thus making NLP a pivotal technology in today's AI-driven era.

In the context of the current thesis, NLP becomes particularly relevant when discussing the foundational model MOLFORMER-XL. As a pretrained AI model, MOLFORMER-XL *infers the structure of molecules from simple representations, making it faster and easier to screen molecules for new applications or to create them from scratch* [13]. The process is akin to the interpretation and analysis of language, where simple text inputs are understood, processed, and used to extract valuable insights, much like NLP operates on human language. In essence, MOLFORMER-XL brings the principles of NLP to the realm of molecular structures, making the relationship between NLP and MOLFORMER-XL an essential aspect of its study.

The subsequent subchapter will delve into tokenization, a fundamental technique of NLP, setting the foundation for a more detailed exploration of the role of NLP in MOLFORMER-XL.

3.1.1 Tokenization

Tokenization is a fundamental process in the handling of written language, particularly when it comes to computational processing. It involves breaking down texts into smaller units, referred to as tokens. These tokens are typically representative of simple words, which are considered the smallest independent units of language. However, tokens can also encompass idioms or hyphenated phrases, such as *user-generated*.

The process of tokenization serves to dissect continuous text into discrete text entities, marking the initial step in any text preprocessing cycle. While the creation of small units is a key aspect, tokenization can also yield entire sentences as output. A rudimentary word tokenizer can be implemented in many languages by segmenting the text at the occurrences of space symbols. This basic approach, however, has its limitations, as it fails to identify words that semantically belong together.

In the context of MOLFORMER-XL, tokenization takes on a unique form. Instead of processing natural language, MOLFORMER-XL deals with chemical language in the form of simplified molecular-input line entry system (SMILES) strings. These strings represent molecules and are tokenized into individual components that represent atoms, bonds, or other chemical features. For instance, a SMILES string like *CCO* would be tokenized into *C*, *C*, and *O*, each representing a carbon or oxygen atom. This process allows MOLFORMER-XL to understand and manipulate the molecular structures represented by SMILES strings.

3.1.2 Vector Space Models (VSM)

In order to facilitate the communication between human beings and computers, several approaches of converting text into a machine readable format have been developed. Some of these methods only represent the statistics of a word, such as one-hot encoding, while others also include the word's context. The vector space model (VSM) is an extension of the one-hot encoding method. Given a set of textual documents, it is possible to create a vocabulary of unique words. Each word is represented by a one-hot encoded vector, which is a vector with the length of the size of the vocabulary with a 1 at the index corresponding to the position of the word in the vocabulary and 0s elsewhere. Furthermore, every document is represented as a vector where each element corresponds to a term (unique word) in the vocabulary. The value at each index is determined by the term frequency, which counts the number of occurrences of each term in the specific document.

In addition to simple term frequency, it's also possible to assign weights to each word according to its relative frequency in the working set of documents. This is often achieved using the term frequency-inverse document frequency method, which assigns lower weights to the more common words. In this case, the weight for a term is calculated by first determining a normalized term frequency, which is the term frequency of the term divided by the sum of all term frequencies in the document. This is then multiplied by the inverse document frequency, which is calculated as the total number of documents in the set of documents divided by the number of documents containing the term. The inverse document frequency is then logarithmically scaled to give lower weights to common words.

In the context of MOLFORMER-XL, the VSM is used to represent molecules. Each unique atom or bond type in the SMILES strings could be considered a *term*, and the frequency of each term in a molecule could be used to create a vector representation of the molecule. However, it's important to note that this approach could result in sparse vectors, as many terms may not appear in a given molecule. Furthermore, the high dimensionality of these vectors could make it difficult to distinguish between different molecules, a problem known as the *curse of dimensionality*, that is tackled in the following section.

3.1.3 Word Embeddings

Word embeddings are a technique used to convert high-dimensional word vectors, such as those produced by one-hot encoding, into a lower-dimensional representation. This concept was initially proposed in [14] as a solution to the *curse of dimensionality*. In the context of word embeddings, the *curse of dimensionality* refers to the difficulty of working with high-dimensional one-hot encoded vectors, especially when the vocabulary size is large. Word embeddings aim to alleviate this problem by mapping each word in the vocabulary to a vector in a lower-dimensional space.

Each word in the vocabulary, is represented by a vector of a predefined dimension. The word

embeddings for all words in the document are stored in an embedding matrix, which acts like a dictionary. Each row of the matrix corresponds to a word in the vocabulary, and the embedding of the word can be retrieved by multiplying the one-hot encoded vector of the word with the respective row. One challenge with using word embeddings is handling out-of-vocabulary words, which are words that are not included in E. There are several methods to create word embeddings, with Word2Vec, GloVe, and fastText being the most popular approaches.

In the context of MOLFORMER-XL, word embeddings are be used to represent the tokens derived from SMILES strings. Each unique atom or bond type could be considered a *word*, and a low-dimensional vector could be assigned to each unique token. This would allow MOLFORMER-XL to capture the semantic relationships between different atoms and bonds, which could potentially improve its ability to generate valid and novel molecules.

3.2 Large Language Models (LLMs)

LLMs are a significant development in NLP and ML, as they contain hundreds of millions to billions of parameters and are trained on extensive text data to understand and generate human language. LLMs use supervised learning, predicting the next word in a sequence, which allows them to generate text similar to their training data. The process of tokenization and word embeddings, discussed earlier, are used to convert text into a format that LLMs can process.

3.2.1 Transformers

Transformers are a type of model architecture used in NLP. Their key innovation is the attention mechanism, which allows the model to focus on the most relevant parts of the input. This, coupled with their highly parallelizable nature, makes them efficient for training on large datasets. However, there are also challenges associated with the transformer approach. The size of a transformer model means it requires significant computational resources for training and storage. Despite these challenges the investment in training a large transformer model can be justified by its versatility. Once trained, it can be fine-tuned for a variety of tasks, potentially saving resources in the long run. Despite the initial cost, the benefits of improved performance and versatility make transformers a promising approach for tasks like molecular representation learning.

In the MOLFORMER-XL project, a transformer model named MOLFORMER is used to learn representations of molecules from chemical SMILES data. This model is a large, general-purpose model trained using unsupervised learning, which is then fine-tuned for the specific task of molecular representation learning. This approach has several advantages. Firstly, the unsupervised pre-training allows the model to learn a broad understanding of molecular structures from a large dataset of 1.1 billion molecules. This leads to improved performance on downstream tasks, as the model

can leverage its broad understanding to make more accurate predictions. The use of a transformer model in MOLFORMER-XL demonstrates the potential of applying NLP techniques to the field of chemistry. The model's ability to learn from large-scale chemical SMILES data and make accurate predictions about various molecular properties provides a powerful tool for drug discovery and material design.

3.2.2 Attention Mechanisms

Attention mechanisms have emerged as a leading approach in NLP, first utilized in machine translation in [15]. The concept of attention is inspired by human cognition, where focus is shifted to the most relevant parts of an information sequence. The degree and location of attention depend on the task and prior knowledge.

The fundamental mechanism of attention involves an RNN, such as an long short-term memory network (LSTM) or a gated recurrent unit (GRU). In this process, each time step leads to the production of hidden states. These hidden states are then transferred to a fully connected FNN, which assigns an attention weight to each hidden state. To ensure these attention weights collectively amount to one, a softmax function is applied to the output of the FNN. After this, the attention weights are multiplied by their corresponding hidden states. The resulting products are then added up to produce a modified hidden state, referred to as the context state. The context state, maintaining the same dimensions as any hidden state from the RNN, is then used for subsequent computations.

In the context of MOLFORMER-XL, attention mechanisms are used to weigh the importance of different tokens in a SMILES string, allowing it to capture the complex relationships between different atoms and bonds in a molecule. This approach enables MOLFORMER-XL to generate valid and novel molecules, demonstrating the power of attention mechanisms in the field of chemistry, which is detailed in the following section.

3.2.3 Area Under the Receiver Operating Characteristic Curve (ROC-AUC)

The receiver operating characteristic (ROC) and the area under the curve (AUC) are crucial components in machine learning, particularly in model performance assessment. These metrics originated from signal detection theory, a field that investigates the ability to discern signals amidst noise. In machine learning, the *signal* is the correct prediction of a class, and the *noise* corresponds to incorrect predictions.

The ROC curve is a graphical representation of a binary classification model's performance across all possible threshold values. The curve is plotted using true positive rate (TPR) on the y-axis against false positive rate (FPR) on the x-axis. Here, the TPR (also known as sensitivity) indicates

the proportion of actual positives that are correctly identified, while the FPR (or 1-specificity) signifies the proportion of actual negatives incorrectly identified as positives. The ROC curve starts at the origin (0,0) and ends at the point (1,1), traversing the area between these points based on the model's performance.

The AUC is a numeric measure that quantifies the total two-dimensional area under the entire ROC curve. It provides an aggregate measure of performance across all possible classification thresholds. The AUC ranges from 0 to 1, where a value of 1 implies that the model has perfect discriminatory ability, and 0.5 suggests that the model's performance is no better than random chance. An AUC less than 0.5 indicates that the model performs worse than random chance, a situation typically resulting from fundamental issues with the model or data.

The importance of ROC-AUC in machine learning is multifold. Firstly, it provides a comprehensive summary of model performance, condensing a multitude of confusion matrices obtained at different thresholds into a single, easily understandable metric. Secondly, it is threshold-independent, implying that it evaluates the model's ability to rank predictions correctly, irrespective of the specific cut-off value chosen. Lastly, ROC-AUC is beneficial when dealing with imbalanced datasets, as it considers both the sensitivity and specificity of the model, rather than focusing solely on accurately predicting the majority class. The ROC-AUC is often used as a comparative metric when selecting the best model from a set of candidates. It helps to identify the model with the most robust performance across different classification thresholds, making it a powerful tool in the machine learning practitioner's arsenal. In conclusion, ROC-AUC is not just a measure of a model's prediction accuracy, but more critically, it reflects the model's capacity to discern the intricate, often overlapping distributions of classes in the feature space.

3.3 Chemoinformatics

Chemoinformatics is a field that combines chemistry, computer science, and information science to solve chemical problems. It involves the use of computational methods to store, manage, analyze, and visualize chemical data. One of the key tools used in chemoinformatics is the SMILES, a notation that allows chemical structures to be represented as text strings. SMILES is a system that uses specific codes to represent atoms, bonds, and other features of a molecule. For example, carbon is represented by the letter *C*, oxygen by *O*, and a double bond is represented by =. So, a molecule of carbon dioxide, which consists of one carbon atom double-bonded to two oxygen atoms, can be represented in SMILES as *C(=O)=O*.

In the MOLFORMER-XL project project, SMILES strings are treated as text data, similar to sentences in a language. Each SMILES string represents a different molecule, just like each sentence represents a different idea. The MOLFORMER-XL model is trained to understand the *language* of these SMILES strings and generate new ones.

The model starts by tokenizing the SMILES strings, breaking them down into smaller tokens, that represent individual atoms, bonds, or other features. These tokens are then converted into numerical vectors using word embeddings, allowing the model to process them. The model then uses a transformer architecture with attention mechanisms to process these vectors. The attention mechanisms allow the model to focus on the most relevant parts of the SMILES string when generating a new molecule. For example, if the model is generating a molecule with a carbon atom, it might pay more attention to tokens representing bonds, as these will determine what other atoms the carbon atom can be connected to. Finally, the model generates a new SMILES string, representing a novel molecule. This string can be converted back into a 3D molecular structure using software tools, allowing chemists to visualize and analyze the new molecule.

4 Use Case 2: MoLFormer-XL

Chapter 4 of this thesis is dedicated to assessing the approach I've taken. Chapter 3 delved into the research aspect of this thesis, laying the groundwork to address the second research question: How do such methodologies fare when applied to genuine descriptive text data in physics research, and how can the performance of generative AI models (LLMs) be enhanced through reinforcement learning?

4.1 Project Overview

Machine learning-based models, especially those driven by ML, offer a promising avenue for accurately predicting molecular properties quickly. This capability is immensely beneficial in areas such as material design and drug discovery. Despite the substantial performance of several supervised ML models, the enormity of chemical space and the scarce availability of property labels pose hurdles to supervised learning. However, unsupervised transformer-based language models, pretrained on vast unlabelled datasets, have recently shown exemplary performance in several NLP tasks. This indicates that employing ML models that learn features autonomously from natural graphs or line notations of molecular structures, such as SMILES, could be a beneficial avenue of exploration.

IBM's MoLFormer, an efficient transformer encoder model, utilizes rotary positional embeddings to train on SMILES sequences of 1.1 billion unlabelled molecules. The MoLFormer model combines linear attention mechanisms with distributed training to derive molecular embeddings. These embeddings have been evaluated against existing benchmarks across several downstream tasks, revealing promising results.

To investigate the possibility of applying the MoLFormer-XL model in other specific domains, such as quantum mechanics and biophysics, IBM Germany granted remote access to an IBM AC922

system, a machine specifically designed for artificial intelligence tasks. The primary challenge in fine-tuning foundation models is the significant computational power requirement. Thus, the focus shifted to exploring the feasibility and performance implications of fine-tuning MoLFormer-XL using only one GPU.

Historically, IBM experiments with MoLFormer-XL utilized 16 V100 GPUs, a costly and scarce configuration in most research centers. Thus, an attempt was made to fine-tune the model using only a single V100 GPU to decrease infrastructure costs. This effort involved restricting the fine-tuning process to use only one of two available GPUs on the system and adjusting the batch sizes for fine-tuning the model. After several attempts with various batch sizes, the V100 GPU utilization was increased above 75% without exhausting the GPU memory.

IBM provided checkpoints for a pre-trained MoLFormer model that had been trained using a dataset of approximately 100M molecules. This dataset combines 10% of the Zinc dataset and 10% of the PubChem molecules dataset used for MoLFormer-XL training. According to the published research paper by IBM, the pre-trained model shows competitive performance on classification and regression benchmarks from MoleculeNet.

The research conducted as part of the project at hand goes beyond experimenting with the IBM MoLFormer-XL foundation model. It pioneers a novel approach of running experiments with very limited computational resources to evaluate the feasibility of utilizing this foundation model when large resources are not available. The insights gained from this study could pave the way for more accessible, cost-efficient deployment of such models in various fields of research.

4.2 Experiments

Taken into consideration IBM work on MoLFormer-XL model, I am exploring in this thesis the possibility of fine tuning the IBM provided model with additional datasets in an efficient way from computational perspective. IBM provided only the checkpoints of a MoLFormer model pre-trained on a dataset of 100M molecules. This dataset combines 10% of Zinc dataset and 10% of PubChem molecules dataset used for MoLFormer-XL training. The accompanying pre-trained model shows competitive performance on classification and regression benchmarks from MoleculeNet according to the published research paper by IBM [13].

Due to the large nature of the combination of the PubChem and Zinc (over 1.1 billion molecules in total) datasets the code expects the data to be in a certain location and format. The details of the of this processing is documented below for each individual dataset.

The code expects both the zinc15(ZINC) and pubchem datasets to be located in “‘./data/“‘ directory of the training directory.

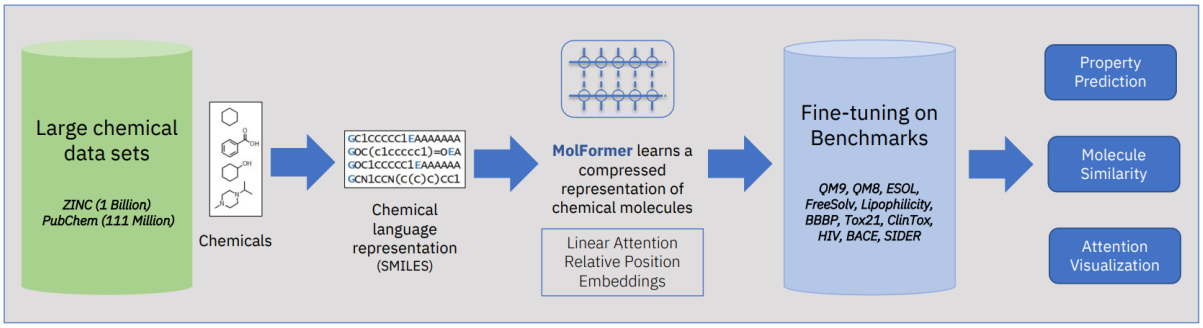

Fig. 4.1: Overview of MoLFormer pipeline.

- Zinc15 itself should be in located "'data/ZINC/"' and is expected to be processed in multiple smi files which contains one smiles string per line
- PubChem should be located in "'data/pubchem/"' and is expected to be processed as a single "CID-SMILES" text file with 2 columns (index and smiles string). IBM took the raw Pubchem dataset and converted every smiles molecule into the canonical form, utilizing rdkit in python, as well as trimmed down the file itself. The dataloader expects Pubchem to be in a converted form and will not run on the raw pubchem file.

Provided IBM MoLFormer-XL checkpoints are for Pytorch framework and only the checkpoints trained on 10% of the IBM dataset over 30000 epochs, therefore will be interesting to get the predictions for classification tasks build around the molecule net tasks. In particularly I will work with 2 additional datasets for finetuning process. In preparation for the training both datasets have been splitted in 70% for training, 28% for testing and 2 percent for validation. The finetuning process will produce training/checkpointing and a .csv file with the validation results contains 4 columns of data:

- Column one contains the validation score for each epoch while column 2 contains the test score for each epoch.
- Column 3 contains the best validation score observed up to that point of fine tuning while column 4 is the test score of the epoch which had the best validation score.

4.2.1 BACE Dataset

This dataset portrays a quantitative (IC50) and qualitative (binary label) binding results for a set of inhibitors of human β-secretase 1(BACE-1) used for regression and classification research problems. A sample output of 10 rows from the datasets is represented bellow, by reading the dataset into a pandas data frame.

In preparation of running all the experiments I had to setup the python environment with the required software packages such as pytorch, cudatoolkit, rdkit, pandas, sckit-learn, apex and scipy (described in the technology section)

	smiles	CID	Class	Unnamed: 3	pIC50	MW	AlogP	HBA	HBD	RB	...	PEOE6 (PEOE6)	PEOE7 (PEOE7)	PEOE8 (PEOE8)	PEOE9 (PEOE9)	PEOE10 (PEOE10)	PEOE11 (PEOE11)	PEOE12 (PEOE12)	PEOE13 (PEOE13)	PEOE14 (PEOE14)	canvasUID
0	Fc1cc(cc(F)c1)C[C@H](NC(=O)[C@@H]1CC[C@](NC(...	BACE_2	1	NaN	8.853872	657.81073	2.6412	5	4	16	...	73.817162	47.171600	365.676940	174.076750	34.923889	7.980170	24.148668	0.000000	24.663788	2
1	S1(=O)(=O)C[C@@H](Cc2cc(O[C@H](COCC)C(F)(F)F)c...	BACE_4	1	NaN	8.698970	591.67828	3.1680	4	3	12	...	56.657166	37.954151	194.353040	202.763350	36.498634	0.980913	8.188327	0.000000	26.385181	4
2	Fc1c2c(ccc1)[C@@]([NH+]=C2N)(C=1C=C(C)C(=O)N(C...	BACE_8	1	NaN	8.612610	477.55200	3.7096	2	0	4	...	42.899986	65.744499	171.121370	146.129900	28.109447	0.000000	-6.106466	13.955495	12.331894	8
3	O=C1N(CCCC1)C(C)(C)[C@@H]1C[C@@H](CCC1)C(=O)N[...	BACE_10	1	NaN	8.602060	562.80573	4.3981	3	3	12	...	93.700790	57.796051	379.486600	151.646620	23.654478	0.000000	24.148668	0.000000	24.663788	10
4	Fc1cc(cc(F)c1)C[C@H](NC(=O)c1cc(cc(c1)C)C(=O)N...	BACE_11	1	NaN	8.522879	594.71179	4.4597	4	3	13	...	46.351440	88.436226	271.730930	178.841130	30.424416	0.000000	23.571255	0.000000	24.663788	11
5	Fc1ccc(cc1)C[C@H](NC(=O)C)[C@H](O)C[NH2+][C@H]...	BACE_21	1	NaN	8.301030	484.62601	2.9008	4	3	9	...	48.077168	45.445873	299.932980	95.216072	34.435734	15.387257	8.188327	0.000000	24.663788	21
6	S(CC[C@H](NC(=O)[C@@H](NC(=O)C)CC(C)C)C(=O)N[C...	BACE_31	1	NaN	8.000000	684.95038	0.7317	8	6	21	...	76.589432	25.401773	552.769960	73.826874	46.405483	7.980170	32.128838	0.000000	32.643959	31
7	Fc1ccc(NC(=O)c2ncc(cc2)C#N)cc1[C@]1(N=C(OC[C@@...	BACE_61	1	NaN	7.552842	371.34079	2.2837	5	1	4	...	0.000000	53.205711	51.124840	105.749890	52.398151	1.587919	9.759030	7.691464	0.000000	61
8	O=C1N(C)C[C@@]1(c1cc(ccc1)-c1cccnc1)c1ccncc1)N	BACE_85	1	NaN	7.221849	343.38190	1.5828	4	0	3	...	34.319988	41.568829	90.833969	129.145870	20.071724	3.271739	7.980170	0.000000	0.000000	85
9	s1cc(cc1)[C@@]1(N=C(N)N(C)C1=O)c1cc(ccc1)-c1cc...	BACE_86	1	NaN	7.221849	348.42151	2.0308	3	0	3	...	34.319988	45.099949	71.636429	118.383720	20.071724	3.271739	7.980170	0.000000	0.000000	86

Fig. 4.2: Quantitative (IC50) and qualitative (binary label) binding results for a set of inhibitors of human β-secretase 1(BACE-1) [16].

Experiment 1.1

For this experiment I have used the BACE dataset with the following training parameters for the MoLFormer-XL model (only the highlighted features have been changed from the recommended parameters of the model publish by IBM.):

- **batch_size 128**
- n_head 12
- n_layer 12
- n_embd 768
- d_dropout 0.1
- dropout 0.1
- lr_start 3e-5
- num_workers 8
- **max_epochs 100**
- num_feats 32
- measure_name Class
- dims 768 768 768 1
- num_classes 2

During the experiment I have observed that the V100 GPUs memory of the system was not fully utilized (only 17GB – approximately 55%) , therefore I took into consideration to increase the batch size (adding more data into the GPU memory at once) and to compute in mor efficient and responsible way.

The experiment ended with a decent ROC-AUC of a value of 0.8533565663976522 with the following results in prediction (only last 5 values are displayed from 100):

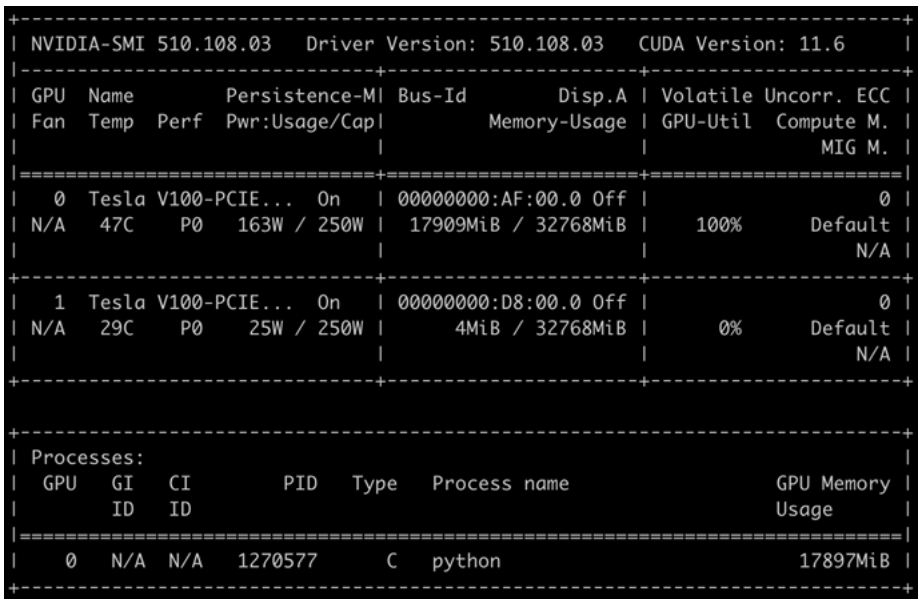

Fig. 4.3: GPU Utilization during the experiment with 128 batch size using BACE dataset.

Epoch	Validation score	Test score	Class	Best Validation score	Score of the epoch
95	10.553.231.239.318.800	12.641.992.568.969.700	10	0.9166368515205725	0.8952677916360969
96	10.423.355.102.539.000	10.785.703.659.057.600	10	0.9166368515205725	0.8952677916360969
97	11.635.489.463.806.100	12.607.325.315.475.400	10	0.9166368515205725	0.8952677916360969
98	11.552.529.335.021.900	13.241.688.013.076.700	10	0.9166368515205725	0.8952677916360969
99	11.406.577.825.546.200	13.274.091.482.162.400	10	0.9166368515205725	0.8952677916360969

Table 4.1: Experiment 1.1 sample results.

Experiment 1.2

For this experiment I have used the BACE dataset with the following training parameters for the MoLFormer-XL model (only the highlighted features have been changed from the recommended parameters of the model publish by IBM.):

- **batch_size 192**
- n_head 12
- n_layer 12
- n_embd 768
- d_dropout 0.1
- dropout 0.1
- lr_start 3e-5
- num_workers 8

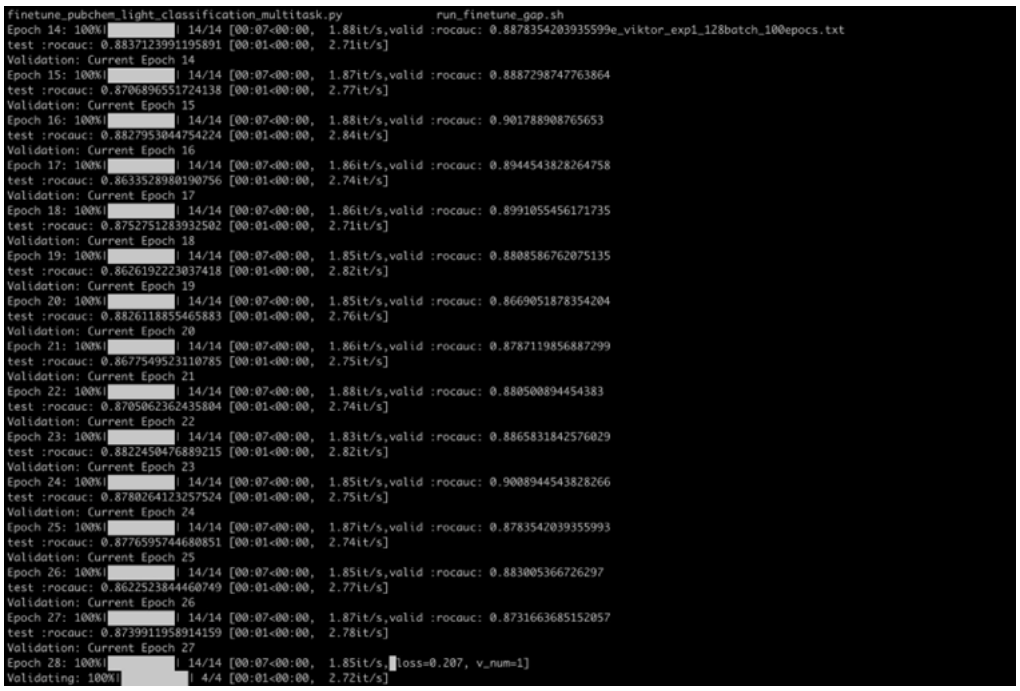

Fig. 4.4: Finetuning process progression over 100 epochs.

- **max_epochs 100**
- num_feats 32
- measure_name Class
- dims 768 768 768 1
- num_classes 2

Initially for this experiment I tried to increase with 2x the batch size (from 128 to 256), however that didn't worked because I have run into a HPU Runtime Error: CUDA out of memory because by doing this I tried to allocate more data into the GPU memory than the physical available memory (GPU 0; 31.75 GiB total capacity; from witch 30.48 GiB was reserved in total by PyTorch). Therefore, I default to 192 batch size.

The experiment ended with a decent ROC-AUC of a value of 0.8747763864042933 with the following results in prediction (only last 5 values are displayed from 100):

Epoch	Validation score	Test score	Class	Best Validation score	Score of the epoch
95	10.553.231.239.318.800	12.641.992.568.969.700	10	0.9166368515205725	0.8952677916360969
96	10.423.355.102.539.000	10.785.703.659.057.600	10	0.9166368515205725	0.8952677916360969
97	11.635.489.463.806.100	12.607.325.315.475.400	10	0.9166368515205725	0.8952677916360969
98	11.552.529.335.021.900	13.241.688.013.076.700	10	0.9166368515205725	0.8952677916360969
99	11.406.577.825.546.200	13.274.091.482.162.400	10	0.9166368515205725	0.8952677916360969

Table 4.2: Experiment 1.2 sample results.

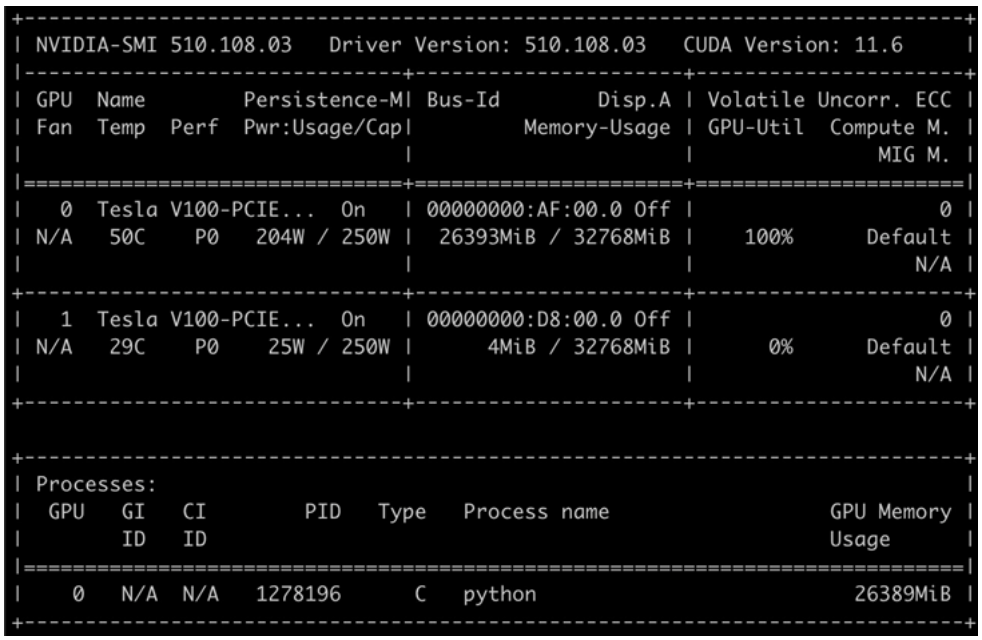

Fig. 4.5: GPU Utilization during the experiment with 192 batch size using BACE dataset.

Experiment 1.3

For this experiment I have used the BACE dataset with the following training parameters for the MoLFormer-XL model (only the highlighted features have been changed from the recommended parameters of the model publish by IBM):

- **batch_size 192**
- n_head 12
- n_embd 768
- d_dropout 0.1
- dropout 0.1
- lr_start 3e-5
- num_workers 8
- **max_epochs 500**
- num_feats 32
- measure_name Class
- dims 768 768 768 1
- num_classes 2

I took into consideration to increase the no of epochs by a factor of 5x to achieve a better finetune model. The experiment ended with a decent ROC-AUC of a value of 0.8487477638640429 with the following results in prediction (only last 5 values are displayed from 500):

Epoch	Validation score	Test score	Class	Best Validation score	Score of the epoch
95	10.553.231.239.318.800	12.641.992.568.969.700	10	0.9166368515205725	0.8952677916360969
96	10.423.355.102.539.000	10.785.703.659.057.600	10	0.9166368515205725	0.8952677916360969
97	11.635.489.463.806.100	12.607.325.315.475.400	10	0.9166368515205725	0.8952677916360969
98	11.552.529.335.021.900	13.241.688.013.076.700	10	0.9166368515205725	0.8952677916360969
99	11.406.577.825.546.200	13.274.091.482.162.400	10	0.9166368515205725	0.8952677916360969

Table 4.3: Experiment 1.3 sample results.

4.2.2 QM9 Dataset

This dataset an enumeration of around 134k stable organic molecules with up to 9 heavy atoms (carbon, oxygen, nitrogen and fluorine). As no filtering is applied, the molecules in this dataset only reflect basic structural constraints. A sample output of 10 rows from the datasets is represented bellow, by reading the dataset into a pandas data frame.

	mol_id	smiles	A	B	C	mu	alpha	homo	lumo	gap	...	zpve	u0	u298	h298	g298	cv	u0_atom	u298_atom	h298_atom	g298_atom
0	gdb_27865	c1([nH]c(=O)nc(n1)O)N	2.05370	1.99411	1.01206	5.4160	63.75	-0.2505	-0.0209	0.2296	...	0.092203	-486.166153	-486.158782	-486.157838	-486.197998	27.954	-1355.509174	-1362.436873	-1369.547805	-1264.861734
1	gdb_85443	CC1NC(=O)NC11CC1	2.57746	1.50208	1.12940	4.0068	75.98	-0.2309	0.0638	0.2947	...	0.160687	-419.248234	-419.239899	-419.238954	-419.280850	32.819	-1840.214696	-1851.870675	-1862.538956	-1710.810417
2	gdb_131846	c1noc(=NCCO)o1	5.83960	0.80010	0.71928	3.0954	66.19	-0.2529	-0.0249	0.2280	...	0.112132	-491.050972	-491.042763	-491.041819	-491.085477	28.863	-1443.764550	-1451.944129	-1460.241053	-1341.775512
3	gdb_10229	CCC(CC)OCC	1.80057	1.68339	0.94450	0.8824	82.47	-0.2409	0.0836	0.3245	...	0.220738	-351.414771	-351.403857	-351.402913	-351.451377	38.416	-2145.089316	-2159.570341	-2173.204857	-1987.819245
4	gdb_101228	CC(CO)N1CC(O)C1	2.85159	0.99923	0.87956	1.8271	79.10	-0.2344	0.0677	0.3020	...	0.193338	-441.450934	-441.441079	-441.440135	-441.485798	36.907	-1978.996469	-1992.364293	-2004.812189	-1830.998471
5	gdb_24268	C1C2OC2C2=C1N=CO2	4.05353	1.61936	1.25976	1.8149	66.12	-0.2344	-0.0108	0.2237	...	0.103792	-436.686266	-436.680267	-436.679323	-436.716654	23.881	-1500.530896	-1509.209973	-1516.913901	-1403.208015
6	gdb_80112	NC1C2N=COC12C#C	2.91235	1.52846	1.28599	1.1571	72.70	-0.2288	-0.0033	0.2255	...	0.112762	-416.753130	-416.745270	-416.744326	-416.785255	30.108	-1530.217720	-1538.617555	-1546.914479	-1427.247258
7	gdb_3297	C#CC1CC2CC12	5.77597	2.11839	1.87636	0.8713	66.18	-0.2541	0.0496	0.3037	...	0.125253	-271.316922	-271.310553	-271.309609	-271.346906	24.758	-1498.073571	-1507.409650	-1515.707829	-1399.515125
8	gdb_116913	C1CC1(C#CCCO)O	4.65489	0.55575	0.54054	2.5024	82.90	-0.2311	0.0417	0.2728	...	0.156629	-422.973602	-422.963257	-422.962313	-423.010549	36.733	-1828.755754	-1839.151068	-1849.819976	-1702.429407
9	gdb_89556	OC1CC2(C1)CC(=O)N2	5.72564	0.84621	0.83943	3.0244	73.06	-0.2508	0.0342	0.2850	...	0.147164	-439.094703	-439.086280	-439.085336	-439.128023	32.368	-1756.143550	-1766.855756	-1776.931668	-1633.918602

Fig. 4.6: Geometric, energetic, electronic and thermodynamic properties of DFT-modelled small molecules [16].

Experiment 2.1

In the second round of the experiments as indicator for the model performance we will use the training loss as a metric to assess how the foundation model fits the training data. That is to say, it assesses the error of the model on the training set (closed to 0 the better the model performance is). For this experiment I have used the QM9 dataset with the following training parameters for the MoLFormer-XL model (only the highlighted features have been changed from the recommended parameters of the model publish by IBM):

- **batch_size 256**
- n_head 12

- n_layer 12
- n_embd 768
- d_dropout 0.1
- dropout 0.1
- lr_start 3e-5
- num_workers 8
- **max_epochs 100**
- num_feats 32
- dims 768 768 768 1

Taken into consideration the nature of the dataset I started with a batch size of 256 and 100 epochs for finetuning. In this experiment I used the training loss as the average of the losses over each batch of training data. Because MoLFormer-XL model is changing over time, the loss over the first batches of an epoch is generally higher than over the last batches. On the other hand, the testing loss for an epoch is computed using the model as it is at the end of the epoch, resulting in a lower loss. The experiment ended with a decent training loss of a value of 0.00454958388581872 with the following results in prediction (only last 5 values are displayed from 100):

Epoch	Validation score	Test score	Class	Best Validation score	Score of the epoch
95	10.553.231.239.318.800	12.641.992.568.969.700	10	0.9166368515205725	0.8952677916360969
96	10.423.355.102.539.000	10.785.703.659.057.600	10	0.9166368515205725	0.8952677916360969
97	11.635.489.463.806.100	12.607.325.315.475.400	10	0.9166368515205725	0.8952677916360969
98	11.552.529.335.021.900	13.241.688.013.076.700	10	0.9166368515205725	0.8952677916360969
99	11.406.577.825.546.200	13.274.091.482.162.400	10	0.9166368515205725	0.8952677916360969

Table 4.4: Experiment 2.1 sample results.

Experiment 2.2

For this experiment I have used the QM9 dataset with the following training parameters for the MoLFormer-XL model (only the highlighted features have been changed from the recommended parameters of the model publish by IBM):

- **batch_size 256**
- n_head 12
- n_layer 12

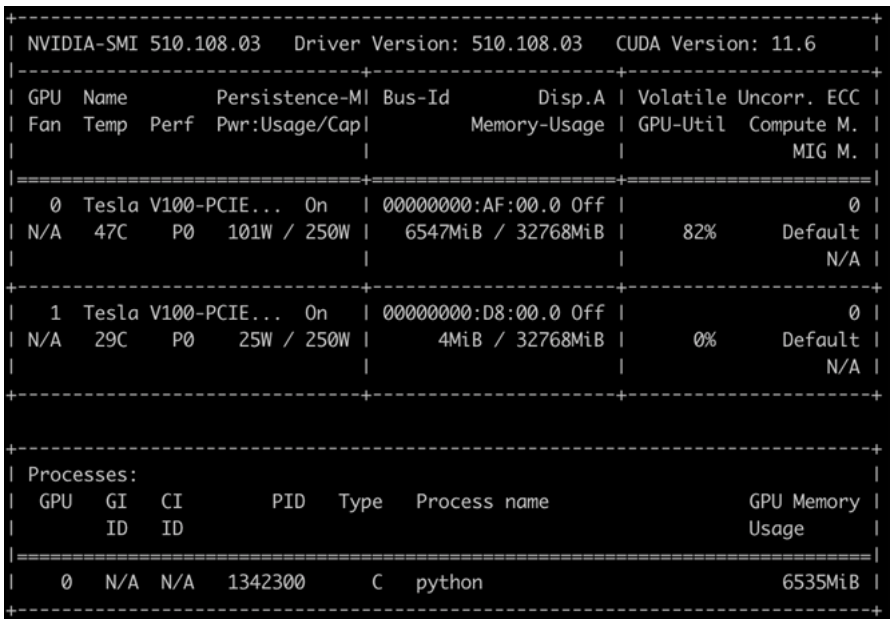

Fig. 4.7: GPU Utilization during the experiment with 256 batch size using QM9 dataset

- n_embd 768
- d_dropout 0.1
- dropout 0.1
- lr_start 3e-5
- num_workers 16
- **max_epochs 500**
- num_feats 32
- dims 768 768 768 1

Because my model seems to achieve very good results in the first experiment I decided to increase no of epochs with 5x (to 500) but not more than this to prevent the overfit of the model.

The experiment ended after 18 hours with a decent training loss of a value of 0.0009311722824349999 with the following results in prediction (only 5 values are displayed from 500):

Epoch	Validation score	Test score	Class	Best Validation score	Score of the epoch
495	0.0011043709237128496	0.001109892618842423	466	0.0004610401811078191	0.000488868507090956
496	0.0016576348571106791	0.00165846711024642	466	0.0004610401811078191	0.000488868507090956
497	0.0005222234758548439	0.0005159794818609953	466	0.0004610401811078191	0.000488868507090956
498	0.0021845398005098104	0.00219011795707047	466	0.0004610401811078191	0.000488868507090956
499	0.0009670979925431311	0.001003631972707808	466	0.0004610401811078191	0.000488868507090956

Table 4.5: Experiment 2.2 sample results.

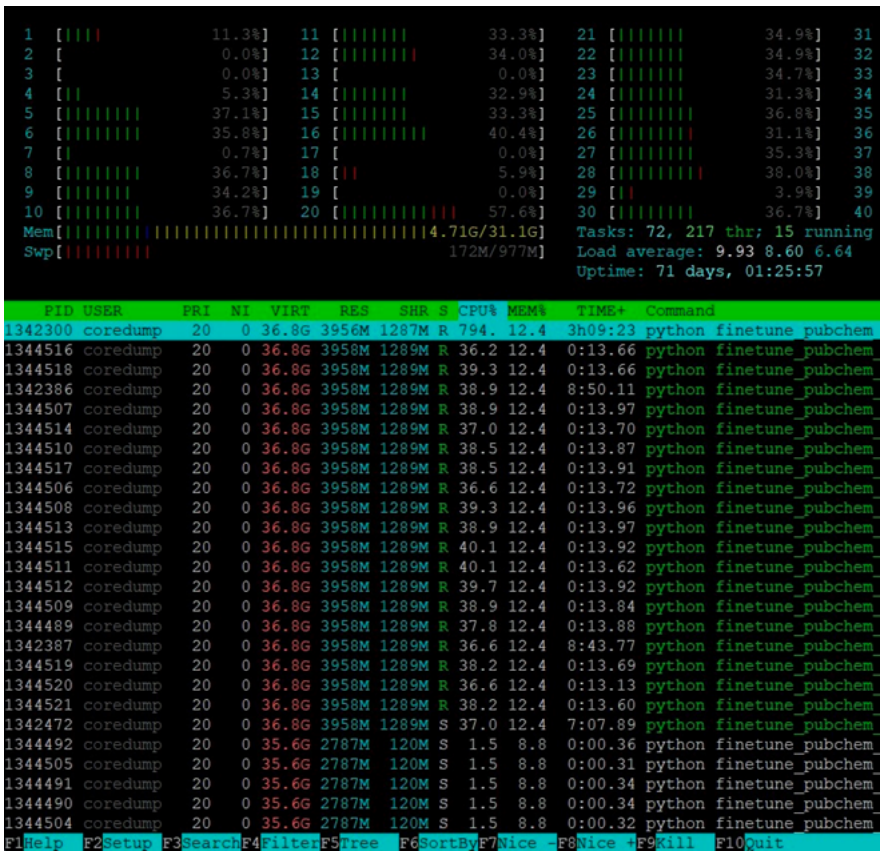

Fig. 4.8: CPU Utilization during the experiment with 256 batch size using QM9 dataset.

Experiment Summary

1. Is clear that for the experiments 1.3 has helped the 5x more training time to achieve a better model with a ROC-AUC value of 0.8487477638640429, while the batch size is at maximum value for the available GPU resources. In a real world, the finetuning process will continue to 20000 epochs. Finetuning of this model on a large no of epochs has not been experimented during the period of the thesis because of need GPU time (for the experiment 1.3 it took 1h and 18 minutes).

2. Similar for the experiment 2.1 and 2.2 we can conclude that running for 500 epochs is increasing the model accuracy however due to the nature of the dataset we can observe that even with a larger batch size of 256 we can't use efficiently the GPU memory and computing power therefore other techniques could be used liked increasing the number of workers.

3. All the experiments have been done using NVIDIA APEX python library in FP16 (mix precision) that allow us to add more data / iteration into the GPU memory compare with FP32 (single precision). This had direct impact in the training time and performance speedup even when conservative mixed precision is used (O1) where only some ops are done in FP16.

4. The fine-tune models can be used only to predict properties of molecules at this point even using a single GPU as demonstrated. In addition, those finetune models can be used to create

a vector representation of the molecules and use them for something like the chemical space visualization.

The main part of the code that was developed for this chapter is provided in Appendix B. The script begins with importing necessary libraries or modules such as $time$, $torch$, $pytorch\_lightning$, $numpy$, etc. It defines a function called RMSELoss which calculates the root mean squared error (RMSE) loss. Next, the main class LightningModule is defined. It takes two parameters: config and tokenizer. This class represents the deep learning model for property prediction. It inherits from pl.LightningModule. Inside the LightningModule class, there are several methods defined:

- $\_init\_$: Initializes the class and sets up the configuration parameters and tokenizer.
- Net: Defines a neural network module for the model.
- lm_layer: Defines a language model layer for the model.
- get_loss: Calculates the loss for the model.
- $on\_save\_checkpoint$ and $on\_load\_checkpoint$: Methods for saving and loading the model checkpoints.
- $configure\_optimizers$: Configures the optimizer for training the model.
- $training\_step$, $validation\_step$, and $validation\_epoch\_end$: Methods for defining the training and validation steps of the model.

The script also includes a PropertyPredictionDataset class which represents the dataset for property prediction. It inherits from torch.utils.data.Dataset. Finally, there is a function $get\_dataset$ for creating an instance of the PropertyPredictionDataset class.

5 Technologies

IBM AC922 System

"The IBM Power System AC922 is the next generation of the IBM POWER processor-based systems, which are designed for deep learning and AI, high-performance data analytics (HPDA), and high-performance computing (HPC). The system is co-designed with OpenPOWER Foundation members and will be deployed at the most powerful supercomputer on the planet by a partnership between IBM, NVIDIA, Mellanox, and others. It provides the most current technologies that are available for HPC and improves the movement of data from memory to graphics processing units (GPUs) and back, which enables faster and lower latency data processing" [17].

This special design system offers a blend of innovative technologies making it a robust AI platform. IBM POWER9 is the first chip with PCIe Gen4 (2x the bandwidth of PCIe Gen3)1, the Power AC922 has PCIe Gen4 and other advanced I/O interconnects including CAPI 2.0, OpenCAPI and NVIDIA® NVLink™. Unlike x86-based servers, on the Power AC922 the NVIDIA® NVLink™ enables CPU to GPU connectivity delivering 5.6x2 the data throughput for today's data-intensive and AI workloads. The Power AC922 supports up to 6 NVIDIA® Tesla V100 GPUs (32GB).

Important characteristics of the system:

- Microprocessors: 2x POWER9 with NVLink CPUs 20 cores
- RAM (memory): 256GB (up to 2TB), 16 DDR4 RDIMM Sockets
- Disk: 2x 1TB SSD
- Processor-to-memory bandwidth: 170 GB/s per socket, 340 GB/s per system
- L2 to L3 cache bandwidth: 7 TB/s on chip bandwidth

Fig. 5.1: IBM AC922 AI System [17].

- GPUs: 2x 32 GB SMX2 NVIDIA Tesla V100 GPUs with NVLink Air-Cooled
- 10GE Ethernet Redundant Network
- Redundant Power supply

NVIDIA V100 32GB SMX2

The NVIDIA® V100 Tensor Core GPU is one of the world's most powerful accelerator for deep learning, machine learning, high-performance computing (HPC), and graphics. By pairing CUDA cores and Tensor Cores within a unified architecture, a single server with V100 GPUs can replace hundreds of commodity CPU [18].

Fig. 5.2: Nvidia V100 32GB SMX2 [18].

Some important features:

- Equipped with 640 Tensor Cores, V100 delivers 130 teraFLOPS (TFLOPS) of deep learning performance.
- NVIDIA Cuda Cores: 5,120
- With a combination of improved raw bandwidth of 900GB/s and higher DRAM utilization efficiency at 95%, V100 delivers 1.5X higher memory bandwidth over Pascal GPUs

- NVIDIA NVLink in V100 delivers 2X higher throughput compared to the previous generation. Up to eight V100 accelerators can be interconnected at up to gigabytes per second (GB/sec) to unleash the highest application performance possible on a single server.

Conclusions

The integration of Artificial Intelligence (AI) into the research process for protein structures has emerged as a promising strategy, with numerous advantages over traditional methodologies. These benefits include enhanced speed and efficiency, improved accuracy, and the ability to integrate multimodal data. However, computational limitations and the requirement for specific software stacks that facilitate the use of foundational models are critical considerations.

Training these models necessitates substantial computational resources, often exceeding those available in typical research facilities, posing a challenge to their integration into production workflows. Nevertheless, the finding demonstrated by the study at hand show that using pre-trained foundational models on domain-specific data can yield satisfactory results, with computational power requirements minimized to the extent of utilizing a single NVIDIA V100 GPU with 32GB of GPU memory.

Despite the fact that some fine-tuning experiments took nearly an entire day - a process that, in a research context, would need to be repeated multiple times - this approach remains feasible. It serves as a valuable asset to augment human research efforts, contrasting with traditional AI models that require extensive human-annotated data for each domain-specific task. Foundational models such as MOLFORMER-XL represent a novel, promising direction for the adoption of AI in research and development processes.

Throughout the conducted experiments, model performance exhibited an acceptable range, as evidenced by the area under the receiver operating characteristic curve (AUC-ROC) values between 0.7 and 0.8. Such performance underscores the necessity for an iterative analysis loop with human researchers for model predictions. As a prospective future direction demanding considerable computational resources, the combination of the foundational model MOLFORMER-XL with a reinforcement learning algorithm, such as the one demonstrated by the implementation of the Varikon Box solver, remains unexplored in this thesis, but could potentially lead to an exceptional model performance (i.e. AUC-ROC of 0.9).

In conclusion, integrating foundational models into the lifecycle of a research product holds significant potential to augment and expedite physics research. These models can equip researchers with potent tools for data generation, simulation, hypothesis testing, and experimental optimization. Nevertheless, it is essential that the results generated by these models undergo rigorous validation against experimental or theoretical benchmarks to ensure their real-world accuracy and reliability in physical systems.

A

List of Acronyms

AI	artificial intelligence
ANNs	artificial neural networks
CRISP-DM	cross-industry standard process for data mining
DM	data mining
DNNs	deep neural networks
DQN	deep Q-learning
DRL	deep reinforcement learning
FNN	feed forward neural networks
GRU	gated recurrent unit
HPC	high-performance computing
KDD	knowledge discovery in databases
LLMs	large language models
LSTM	long short-term memory network
MCTS	Monte Carlo tree search
ML	machine learning
NLP	natural language processing
RL	reinforcement learning
RNN	recurrent neural networks
SMILES	simplified molecular-input line entry system
VSM	vector space model

Bibliography

[1] R. Feldman and J. Sanger, *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.

[2] N. Indurkhya and F. J. Damerau, *Handbook of natural language processing*. Chapman and Hall/CRC, 2010.

[3] S. Marsland, *Machine learning: an algorithmic perspective*. CRC press, 2015.

[4] M. Allahyari, S. Pouriyeh, M. Assefi, *et al.*, *A brief survey of text mining: Classification, clustering and extraction techniques*, 2017. arXiv: 1707.02919 [cs.CL].

[5] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, "Knowledge discovery in databases: An overview," *AI magazine*, vol. 13, no. 3, pp. 57–57, 1992.

[6] P. Chapman, J. Clinton, R. Kerber, *et al.*, "Crisp-dm 1.0," *CRISP-DM Consortium*, vol. 76, no. 3, 2000.

[7] B. Marr, *Artificial intelligence in practice: how 50 successful companies used AI and machine learning to solve problems*. John Wiley & Sons, 2019.

[8] M. Tom, "Mitchell: Machine learning," *1997 Burr Ridge*, vol. 45, no. 37, pp. 870–877, 1997.

[9] D. Fumo, "Types of machine learning algorithms you should know," *towards data science*, vol. 15, 2017.

[10] P. Madan and S. Madhavan, "An introduction to deep learning," *IBM Developer*, vol. 3, 2020.

[11] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete event dynamic systems*, vol. 13, no. 1-2, pp. 41–77, 2003.

[12] JimPuzzles, *Varikon Box 3x3x3*, https://www.cs.brandeis.edu/~storer/JimPuzzles/ZPAGES/zzzVarikonBox3x3x3.html, Accessed: 29-11-2022.

[13] J. Ross, B. Belgodere, V. Chenthamarakshan, I. Padhi, Y. Mroueh, and P. Das, "Large-scale chemical language representations capture molecular structure and properties," *Nature Machine Intelligence*, vol. 4, no. 12, pp. 1256–1264, 2022.

[14] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," *Advances in neural information processing systems*, vol. 13, 2000.

[15] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL].

[16] Z. Wu, B. Ramsundar, E. N. Feinberg, *et al.*, "Moleculenet: A benchmark for molecular machine learning," *Chemical science*, vol. 9, no. 2, pp. 513–530, 2018.

[17] R. Nohria and G. Santos, “Ibm power system ac922–technical overview and introduction,” REDP-5494-00, IBM Redbooks, Tech. Rep., 2018.

[18] 2020 NVIDIA Corporation, *NVIDIA V100 TENSOR CORE GPU*, https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf, Accessed: 29-03-2023.