A High-throughput pose selection method for extreme scale virtual screening in drug discovery

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Author: **Gianmarco Accordi**

Student ID: 940437
Advisor: Prof. Gianluca Palermo
Co-advisors: Davide Gadioli, Alfonso Gautieri
Academic Year: 2021-22

Abstract

The human capacity of treating illness is one of the mechanisms that have led our species to an overall increase in life quality and an increase in the human population. The drug discovery process aims at finding suitable treatments for diseases. This process is usually carried out with in vitro testing of compounds against a target protein. In this way, the process can be very long and costly. It requires time, human efforts, and chemical components. With the advent of supercomputers, arises the idea of using computer-generated models to get a hint on the most promising compounds. This generated a new field of research on virtual screening pipeline, which is a pipeline of compounds screening against a target protein, done virtually. Since a large dataset of compounds exists it becomes necessary to scale an HPC infrastructure to analyze each compound against the disease. Nowadays high-performance computing environment is facing an increasing challenge in trying to continuously speed up the computation: Moore's law decline makes it impossible to increase the computational power by relying only on components miniaturization. These premises created the general interest in pursuing new techniques, that enable us to be more efficient while being faster. So we need to keep in mind the balance that exists between memory usage and computation requirements. The outbreak of the recent pandemic has shown us how the existence of a fast and reliable process of molecular docking for drug discovery [1], could lead to a faster response when fighting these such emergencies in the next future [2]. This document has the objective of proposing a way in which we can increase the throughput of a molecular docking simulation as much as possible, by analyzing which part of the pipeline can be accelerated. The techniques used in this thesis to accelerate and application comes from approximated computing field. In the end, we will apply the result of the analysis on the pipeline of a CADD: LiGen.

Keywords: Molecular Docking, Drug Discovery, XSCORE, Autodok, Autogrid, Approximate Computing, Scoring Function, GPU, HPC, High performance Computing, Virtual Screening, Ligand, Poses, Receptors, Precision Scaling, Memoization, LiGen

Abstract in lingua italiana

La capacità della razza umana nel curare le malattie è uno dei motivi che ha permesso alla nostra specie di aumentare la sua aspettativa di vita, e di conseguenza anche di aumentare la sua popolazione. La scoperta di cure per le malattie viene solitamenta fatta eseguendo test in vitro, in cui vengono analizzati gli effetti di vari composti chimici sui recettori di alcune malattie. Ques'ultimo è però un processo molto lungo e costoso. Richiede tempo, fatica, e reagenti chimici. Con l'avvento dei supercomputer, è nata l'idea di usare dei model computerizzati per ottenere degli indizzi su quali potessero essere i composti chimici di maggiore interesse per fare test in vitro. Ciò ha portato alla nascita di molti studi sulle virtual screening pipeline, cioè pipeline di analisi degli effetti di vari composti chimici su diverse malattie. Siccome al giorno d'oggi esistono grandi raccolte di composti chimici conosciuti dalla scienza, è necessario progettare una struttura HPC per analizzare ogni composto contro la malattia obbiettivo dell'analisi. A questo punto i ricercatori nell'ambito del HPC devono affrontare una sfida sempre maggiore, nell'accelerare la computazione: il declino della legge di Moore rende impossibile l'aumento della potenza computazione basandosi solo sulla miniaturizzazione delle componenti. Ed è proprio per questo che i ricercatori hanno iniziato a perseguire nuove strade, per essere più veloci ed efficenti nella computazione. Bisogna però sempre tenere in considerazione il tradeoff che esiste tra l'utilizzo della memoria e la potenza di calcolo richiesta. Lo scoppio della recente pandemia ha mostrato come una più vloce virtual screeing pipeline [1] possa essere di aiuto nel combattere emergenze future simili a questa [2]. Questa tesi ha come obbiettivo quello di proporre un metodo che può essere utilizzato per aumentare l'output di una simulazione di molecular docking il più possibile, attraverso un'analisi delle componenti della pipeline che possono essere accelerate. Le tecniche utilizzate per accelerare la pipeline sono prese dal campo dell'approximated computing. Alla fine analizzeremo l'impatto di queste tecniche su un CADD: LiGen.

Parole chiave: Molecular Docking, Drug Discovery, XSCORE, Autodok, Autogrid, Approximate Computing, Scoring Function, GPU, HPC, High performance Computing, Virtual Screening, Ligand, Poses, Receptors, Precision Scaling, Memoization, LiGen

Contents

1 | Introduction

In all of human history, we have pursued the objective of increasing our quality of life. Breakthrough discoveries in the medical field, have led our lifespan of decades. When humanity encounters a new disease, it takes its capacity to find a treatment for this new illness. In the past centuries, the discovery of new treatments has been based on the human capacity of finding new compounds able to inhibit the effects of the disease. This process is called drug discovery, which is largely used by the pharmaceutical industry. Before the advent of computers, researchers have to create new compounds and test them in vitro against the target disease, to discover the compound's effects on the disease. This is process relies on human intuition in finding the desired compound, and so it can be very long and costly because, each compound under analysis, requires time, human efforts, and chemical components. A first attempt to automatize this process has been carried out through high-throughput screening techniques, which used machines to replace researchers. Today machines are faster and can test a larger number of compounds [3]. The digitalization process in act nowadays has targeted also the pharmaceutical industry, which has aggregated its knowledge into a big dataset of compounds [4]. Nowadays the limit in the number of tested compounds is not on the number of compounds that can be tested in vitro, but rather on the throughput of a computer-generated model which anticipates the in vitro testing. We can use a computer-generated model for the analysis of the big ligand's dataset, and we can identify which compounds should be passed to the in vitro testing. The simulation of the interactions between molecules (as a compound) with a computer model, goes into the research field of molecular docking algorithms. With this algorithm, researchers can find out how a compound interacts with another compound. In this way, the drug discovery process is partially digitalized. Because it relies on the results obtained by a computer model to get hints on the most promising compounds that should be passed to later stages of the process, for further analysis. The virtual screening pipeline requires as input a set of compounds, which will be digitally tested on the receptor of a disease, in search of the compounds that can inhibit such a disease. HPC becomes important to use the whole super-computing infrastructure. The complexity of an algorithm is evaluated as a function of the input's size. To increase the throughput of a virtual screening pipeline. To get a high throughput we should feed our application with enough data, so as not to remain idle. So the input's size should be increased, in this way the complexity is increased, and a higher complexity means a higher time to solution. In the following approximated co muting technique are used to reduce the complexity of virtual screening pipeline algorithms.

Molecular docking algorithms can be used to understand how a compound can inhibit a disease, then scoring is used to evaluate which ligands should be further tested. Having at disposal big datasets of compounds can be combined with big computational power, to deploy a high-throughput virtual screen pipeline, which is used to speed up the drug discovery process. The evaluation of the most promising compounds has to be done with a well-defined metric, so to each compound, the pipeline assigns a score. The compounds with the highest score are selected for later stages of the pipeline. The purpose of this thesis is to pursue new techniques which help a computer-generated model to increase its throughput, thus increasing the number of tested compounds against a targeted illness. Usually, for an application, a small time-to-solution is an important requirement, for which a lot of effort is put on. When dealing with a fast-spreading disease, as in the recent pandemic, is preferable to have a fast response time. The response's objective is to find suitable treatments for a disease outbreak [1], thus is important to continue studies in the urgent computing field.

The original drug discovery process, where scientist manually tests each compound against a disease, makes use of the researchers' ability to foresee which are the most promising compounds against a disease. This is already an approximation because we are not analyzing the whole input space of compounds. The input space would be too big for an exhaustive analysis, instead, the original drug discovery process relies on domain-expert knowledge. During the last decades, the requirement of increasing an application's throughput has been met thanks to the computers' components' miniaturization, which has increased the computational power, while decreasing the computational cost. The decline of Moore's law [5] has led the researchers to think differently to meet the requirements. This document studies the possibility of increasing the number of compounds tested in the same amount of time, by using approximation techniques. As an alternative, we can use a computer-generated model, which simulates the real-world interactions between a compound and a disease. But since the real world is continuous while the digitalized world is discrete, some other approximations have to be applied, to make the virtual simulation feasible and to deploy a molecular docking algorithm. Today is easy to access big datasets of compounds (for example PDBBind [6]), which makes possible an exhaustive test of compounds against a disease. Based on the computer-generated model used, the computation can be very long and complex. So other approximations can be

introduced to increase the throughput of these models, and also increase the errors in the solution. Approximated computing allows us to consider a trade-off between the needed models' throughput and the results' accuracies, required by the application.
These approximations of interest come from the approximated computing research field. Approximated computing analyzes the impact of approximations techniques on different applications, based on the application's sensibility to the results' errors. Approximate an application requires gaining deeper knowledge about its working mechanisms, which is then used to find approximations opportunities. The solution proposed in this thesis is to analyze a virtual screening pipeline in seek of approximations opportunities, and then different approximations strategies will be applied to increase the throughput. The approximations strategies used in this thesis, are used to reduce the input space to make it finite, and so to make possible the pre-computation of some data, required during the compounds' evaluation. The pre-computation has to be carried on once for each protein under analysis. Then a filtering stage which requires as input the docked poses and the pre-computed data (used to evaluate the incoming poses), can be used in different approaches: on one hand, it can be used as a scoring function, on the other hand, it can be used as a filter between the docking and the scoring stage.
The necessity of increasing the throughput of an extreme scale virtual screening pipeline is the reason behind the efforts made during the thesis. So the core of the thesis is in the explanation of our approach to defining a solution, along with an analysis of the solution's implementation. The understanding of this thesis's content requires the introduction of some base chemistry and computational theory concepts Chapter 2. Once these concepts are clear a state-of-art analysis is required Chapter 3, to better comprehend the requirements of the drug discovery process, and the available approach in the literature. The proposed methodology to solve the problem is now introduced Chapter 4, since the problem scope of this thesis will be well defined. And in the last part, this document reviews the obtained results Chapter 5, with a focus on the advantages someone can gain when using our proposed methodology. In support of the results, also a real-world use case has been analyzed Section 5.6. Finally, Chapter 6 concludes the thesis.

2 | Background

This thesis seeks new techniques which can increase the throughput of an application. The application of most interest in this thesis is drug discovery. Since Approximated computed applied to the drug discovery process will be the most important topic for the rest of this document, it is essential the introduction of some keywords and concepts. The understanding of topics covered in this thesis requires the assimilation of certain concepts, that range from basic chemistry to more advanced notions of computational complexity. This chapter will give a better definition of such concepts.

2.1. Ligand, Poses, Receptor

The term Ligand comes from the Latin word *ligare* (which means to bind), and they can be anions, cations, or neutral molecules, but they are usually composed of tens of atoms. Figure 2.1 shows a ligand representation when loaded by PyMol, which is molecular visualization system [7]. Ligand can change their conformation, as Figure 2.2 shows. Figure (a) shows a ligand pose, which is a ligand's conformation in space, with a well-defined spatial transformation. Figure (b) shows the same ligand with a different pose. While figure (c) shows two ligands' poses overlapped. If a ligand is considered as a rigid body then it will have 6 degrees of freedom (3 for translation and 3 for rotations). Otherwise, if the ligand is considered to be flexible its degrees of freedom are $6+r$, where r is the number of rotatable bonds in it. Rotatable bonds, also called rotors, are any non-ring bonds, bounded to non-terminal heavy atoms. As the name suggests rotatable bonds can rotate, thus changing the pose conformation. The difference between the same ligand conformation is usually measured with the RMSD or Roor-Mean-Square Deviation, which is the average distance between the same atom position in the different conformations. Their conformation can change in space based on the environment and the different forces in action. They become of great interest when they are considered in pair with a Receptor. Ligands can bind with receptors, which are the target of a drug discovery process. Usually, a disease is identified by the receptor protein, for which the process aims at finding a ligand that inhibits the protein activity. Receptors are a class of proteins that can bind

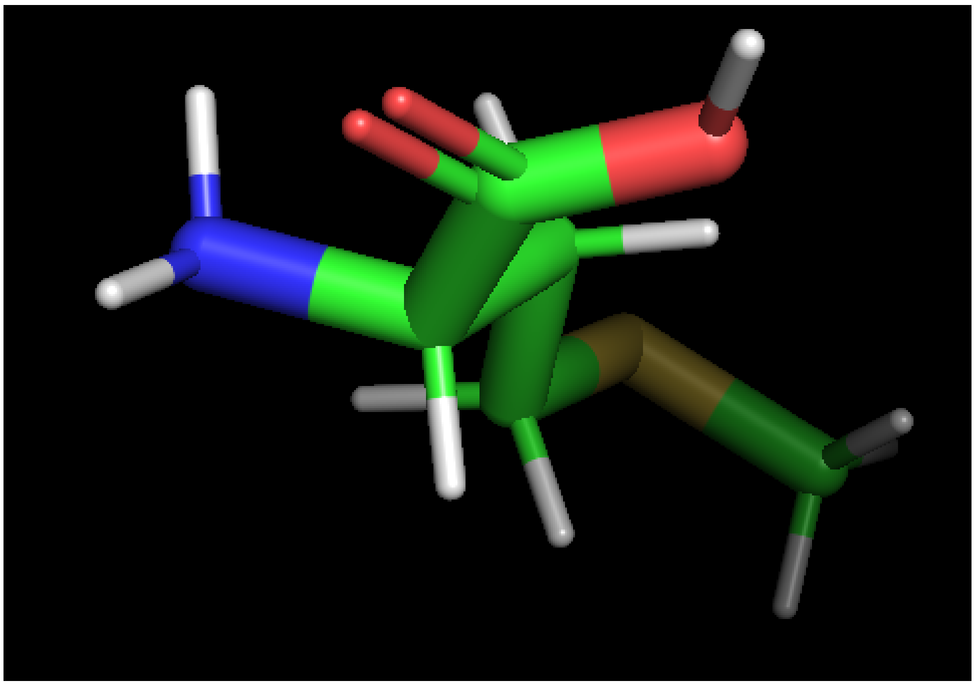


Figure 2.1: A representation of a ligand taken from PyMol.

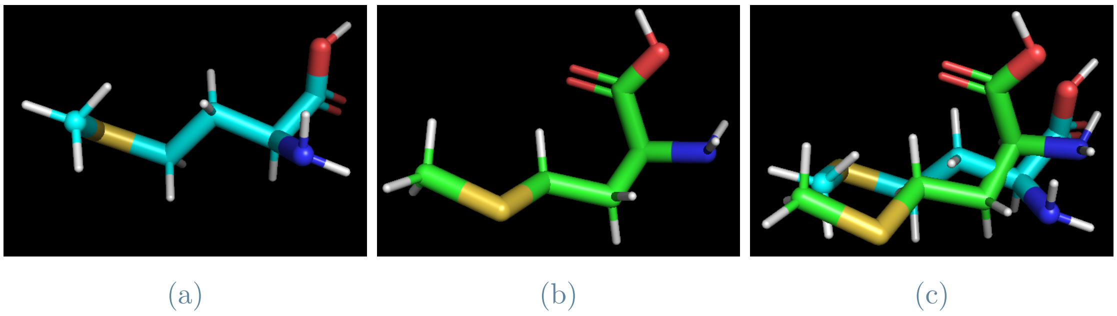


Figure 2.2: Different ligands' poses.

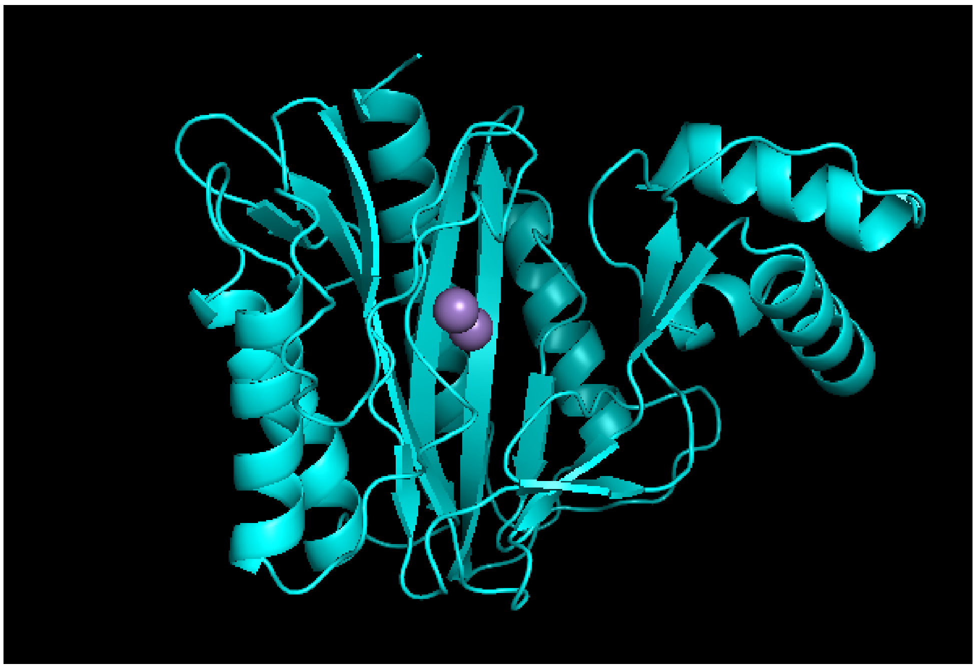


Figure 2.3: Receptor 00007_1WKM as seen by PyMol.

to ligand molecules. A receptor visualization is reported in Figure 2.3. Some portions of the receptor's surface are more suitable for binding with a ligand: these are called binding sites. Based on where a ligand binds to a receptor, it changes its conformation to form a stronger bond with the receptor. Since a receptor can have multiple binding sites, in which a ligand can be found with different conformations, we call each specific conformation a pose. Receptor binding sites are also referred to as pockets of the receptor. An example of a ligand docked to a receptor is reported in Figure 2.1. These pairs of ligand-protein are usually extracted, for analysis, from PDBBind [6]: a collection of experimental binding between proteins and ligands. For each couple, PDBBind also provides the crystal pose. The crystal pose is the reference pose of the ligand upon binding with the paired receptor. The crystal is determined through X-ray Crystallography, as the name suggests.

2.2. Drug Discovery

Drug Discovery is the process of finding new treatments for diseases. The process began because there is a disease without any suitable medication. Usually, the treatment of a disease coincides with the inhibition of a protein, which will be the receptor. The outcome of drug discovery will be a ligand, a small molecule able to inhibit the receptor under study. This is a long and capital-intensive process, as Figure 2.5 shows. The process starts with basic research and lead discovery stages, which are important for the identification of

Figure 2.4: Receptor 00007_1WKM which forms a bond with the ligand in Figure 2.1.

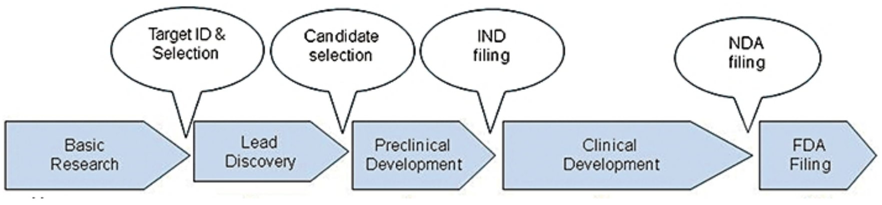


Figure 2.5: Drug discovery stages, taken from [8]. IND stands for Investigation New Drug, while NDA stands for New Drug Application.

compounds that can potentially inhibit the activity of the disease. The best candidates will pass later on to the preclinical and then clinical development. If the drug can pass all these stages then is marked as a drug. Figure 2.5 also shows how on average the duration of a drug discovery process is longer than 10 years. Since the drug discovery process is very important for the pharmaceutical industry has a great interest in the drug discovery field. Over time different techniques have been used to discover new drugs:

- *HTS* High-Throughput Screening which automates the process of testing a big dataset of ligand against a receptor in vitro, accelerating the work done by a human researcher [8];
- *VHTS* Virtual High-Throughput Screening, is similar to HTS as the name suggests, but in this case, computer-generated models are used to evaluate how a ligand bind with a receptor [8];
- *FBDD* Fragment-based Lead Discovery, uses smaller organic compounds, which are then combined to obtain a compound with the desired effect against the receptor [8].

Among them, one of the most promising techniques in the drug discovery process is the VHTS. VHTS requires computer-generated models, that are constructed starting from Molecular Docking's knowledge.

2.3. Virtual Screening

Nowadays, the pharmaceutical industry has collected a big dataset of compounds. Compounds are used to inhibit disease, so testing each compound against each disease is impossible. A more feasible option is to use a supercomputer to analyze which compounds have a relevant effect on a particular disease. The analysis task is very complex also for a computer: it requires time. The recent pandemic has shown us that a fast development in treatment against new diseases can help in increasing the quality of the response to such an event. Virtual Screening is the experimental screening of a large data-set of compounds against a target receptor, intending to identify the lead compounds. Lead compounds are the ligands that are most likely to form a stable complex with the receptor. Thus they require further investigation. The key idea is that we want to automatize the whole process as much as possible. Compounds are tested computationally, to reduce compounds that have to be physically screened in the laboratory by the researcher. Crucial in this virtual screening pipeline are two stages: Sampling and Scoring, which can be seen in Figure 2.6. Starting from the ligand and the protein the pipeline uses the sampling stage to produce output ligands' conformations upon binding with the receptor. These output

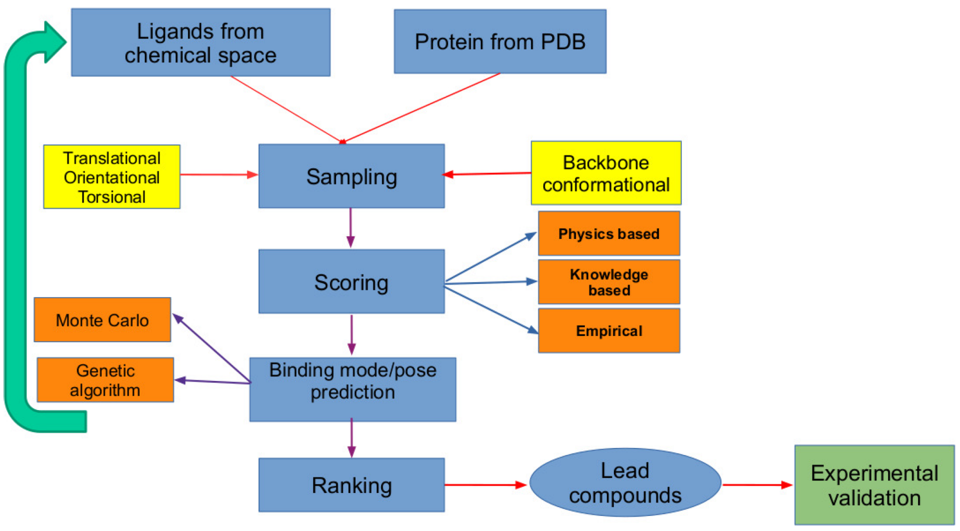


Figure 2.6: Virtual Screening stages representation taken from [9].

poses are scored. Based on the score a rank of the poses is produced, from which the leads compounds are extracted. In the literature, not only virtual screening exists, but different screening techniques can be found:

- *High throughput* pharmaceutical companies use a large dataset of compounds to perform an exhaustive search on the whole combination ligand-receptor available [8];
- *Focused Screen* compounds that have been already found to be active against specific receptors are used as a starting point to design new drugs along the pipeline [8];
- *Fragment Screen* small compounds are used as a building block for large molecules to be tested [8];
- *Structural Aided Drug Design* the design of new molecules starts from the analysis of the crystal structure [8];
- *Virtual Screening* combining the crystal structure along with a known ligand to build other compounds further on.

The most important stages in a virtual screening pipeline are sampling and scoring. Sampling analyzes the whole search space for all the possible orientations and conformations of the ligand paired with the receptor. The search space is too big for an exhaustive search. A value of energy is associated with each new conformation of the ligand inside the receptor's binding site. This energy evaluates the stability of the ligand-protein complex. Virtual screening programs use heuristic techniques to find the local minima of the energy associated with the pose. These techniques range from Monte-Carlo to genetic algorithms.

Scoring happens right after Sampling. The selected ligands' conformations have to be ranked. From the rank, the lead compounds will be chosen. The poses' rank is defined by the score computed with a scoring function. Different types of scoring functions exist in the literature [10]:

- *Force Field* computes the score based on the intramolecular interactions among ligand and receptor, like Van der Waals and electrostatic interactions;
- *Empirical* different binding factors are taken into consideration. Each of these factors is associated with a coefficient. They are combined to fit in multiple linear regression;
- *Knowledge Based* it solves the problem by using a statistical potential function. It gains knowledge from a set of protein-ligand from the PDB database.
- *Machine Learning* the scoring function is inherited from the data.

A visual representation of scoring functions is provided by Figure 2.7. The output of the docking stage is evaluated by the scoring stage, which will select the favorite complex. This evaluation is done using a scoring function, which can be of different types. They are classified based on which factors of atoms' interactions they consider and the weight they put on each factor. Usually, scoring functions are additive, which means that they sum up the considered factors. In some cases, consensus scoring is implemented: multiple scoring functions are computed and later used in a voting system.
Lead compounds are then used as a hint for in vitro analysis, a later step in the drug discovery pipeline.

2.4. Molecular Docking

Molecular Docking is used to predict the best pose in which a ligand can be bound to a receptor to form a stable complex. Molecular Docking can be considered as a *lock-and-key*: the lock is the receptor (more precisely the binding site), while the ligand is the key. It is used to predict the best orientation of the ligand in the binding site of the receptor. Thus molecular docking plays an important role in the drug discovery process. Different molecular docking approaches exists [11]:

- *Rigid Docking* ligand and receptor conformations does not change upon binding [11];
- *Flexible Docking* the involved molecule are free to change their conformation when docked [11];
- *Semi-flexible Docking* ligand and target are allowed to change their structure within

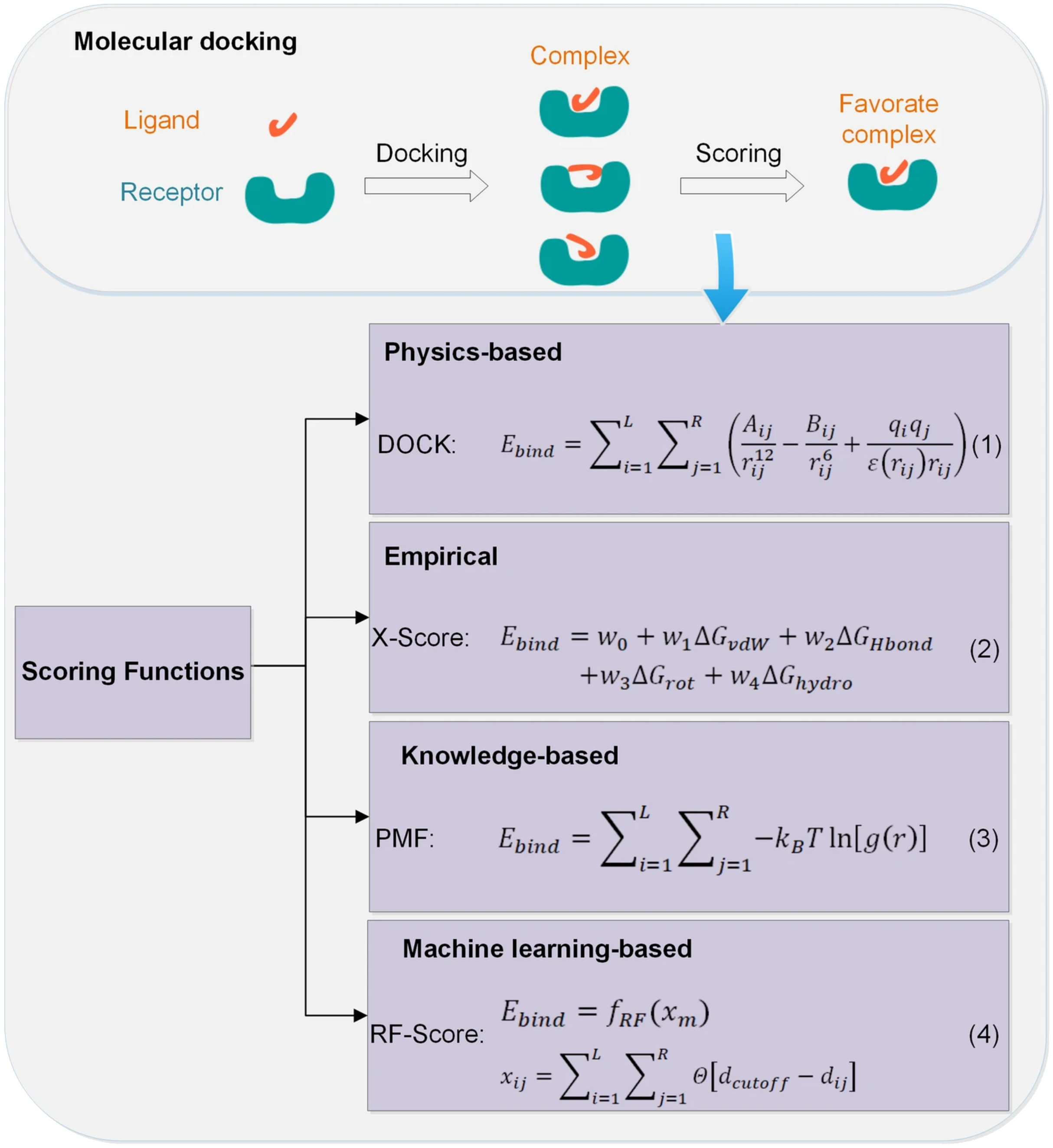

Molecular docking
Ligand
Receptor
Docking
Complex
Scoring
Favorate complex
Scoring Functions
Physics-based
DOCK: $E_{bind} = \sum_{i=1}^{L}\sum_{j=1}^{R}\left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + \frac{q_i q_j}{\varepsilon(r_{ij})r_{ij}}\right)$ (1)
Empirical
X-Score: $E_{bind} = w_0 + w_1\Delta G_{vdW} + w_2\Delta G_{Hbond} + w_3\Delta G_{rot} + w_4\Delta G_{hydro}$ (2)
Knowledge-based
PMF: $E_{bind} = \sum_{i=1}^{L}\sum_{j=1}^{R} -k_B T \ln[g(r)]$ (3)
Machine learning-based
RF-Score: $E_{bind} = f_{RF}(x_m)$, $x_{ij} = \sum_{i=1}^{L}\sum_{j=1}^{R}\Theta[d_{cutoff} - d_{ij}]$ (4)

Figure 2.7: Molecular docking and scoring functions types, taken from [10].

a certain range [11].

The main difference among them is in how flexible they consider the structure of the ligand and of the receptor. More flexible the structures are considered to be, the higher the computation power required to perform molecular docking [11].

2.5. HPC

From Section 2.2 is clear that the drug discovery process is long and costly. So for the pharmaceutical industry, the cost of using a supercomputer infrastructure that runs a computer-generated model to restrict the number of compounds which should be passed to the clinical and preclinical stages of the drug discovery process, is very convenient, with respect to the whole in vitro drug discovery process. High-Performance Computing uses computer clusters to solve complex computational tasks. Docking ligands with a receptor is an embarrassedly parallel task: each ligand can be docked with the receptor independently from the other ligands. HPC infrastructures are composed of a big number of nodes. Nodes are equal, usually composed of a CPU, and the best supercomputers in the world are usually equipped with some accelerators, like multiple GPUs per node [12]. The virtual screening task can thus be decomposed: ligands can be docked against the receptor at the same moment on different nodes.
Advances in the High-Performance Computing infrastructure increase the computational power, and thus the throughput of an application executed on them. So for example Molecular Docking becomes faster to be executed on an HPC infrastructure: in early times docking was performed using a rigid structure for the ligands, leading to big variance in the binding site conformations [3], nonetheless, this leads to some successful drug design example [13]. Even if computation was quite inaccurate, only a small amount of compounds have been considered, since the computational power limit of that time.
Thanks to the computational progress we have done up to now, nowadays is it possible to score a large number of ligands against a single target in a reasonable time frame. This is due to the level of parallelism used [14]: we can decide to parallelize different stages of the Molecular Docking Pipeline, we can parallelize the analysis of multiple ligand-protein pairs on different computing units, we can parallelize the evaluation of a single pair ligand-protein, by divide and conquer on the single atoms. To support and speed up the development of these architectures, multiple technology stacks exist, like MPI [15] and CUDA [16].
Great results have been obtained using this approach: for example on the Summit IBM supercomputer, researchers have been able to perform exhaustive docking of one billion

ligands in under 24 hours [3].

2.6. Complexity

The quality of an algorithm can be evaluated in terms of memory requirement and computational complexity. In this document, HPC infrastructure is used to resolve the molecular docking problem. The complexity of an algorithm is evaluated from a function of its inputs $f(n)$. For example one can say that the worst-case complexity of an algorithm is $f(n) = n^3$. In that case, the time requires by the algorithm to resolve the problem grows as n^3 where n is the size of the input. Computational complexity grows following different functions of the input, based on the algorithm used. As the complexity grows also the time required to resolve the problem grows with it. The bottleneck of our application is the complexity of the algorithm, used to solve the molecular docking problem, rather than the memory complexity of our application.
Docking a ligand on a receptor, using a semi-flexible docking algorithm, requires a lot of computation based on the ligand's atom number and on the receptor's pocket size. Because the highest the number of atoms involved during the docking, the highest the degrees of freedom, which will increase a lot the solution space the virtual screening simulation has to explore.

2.7. Approximated Computing

Increasing the throughput of a virtual screening task is a difficult task. The computational complexity of the molecular docking algorithm becomes a factor that cannot be reduced any longer, after a certain point. Thus other approaches should be considered: approximate computing is used to trade computational complexity, thus computing time, with the accuracy of the results, as shown in Figure 2.8 [17]. An example of approximated computing is loop perforation: some iterations of a loop are skipped, as the name suggests. Before applying approximate computing to a computation one should analyze how much the application is error-resilient. Approximated computing can be introduced at multiple layers of the system stack: architectures, programming models and algorithms can benefit from approximated computing. Approximated computing has been applied to a wide range of fields, from machine learning to graphic application. Similarly, approximated computing can be applied to the molecular docking problem. The following chapters will show how approximated computing is not able to solve all the problems of molecular docking algorithms: precise analysis and definitions of metrics in the output quality should be considered. Metrics help in understanding the level of approximation

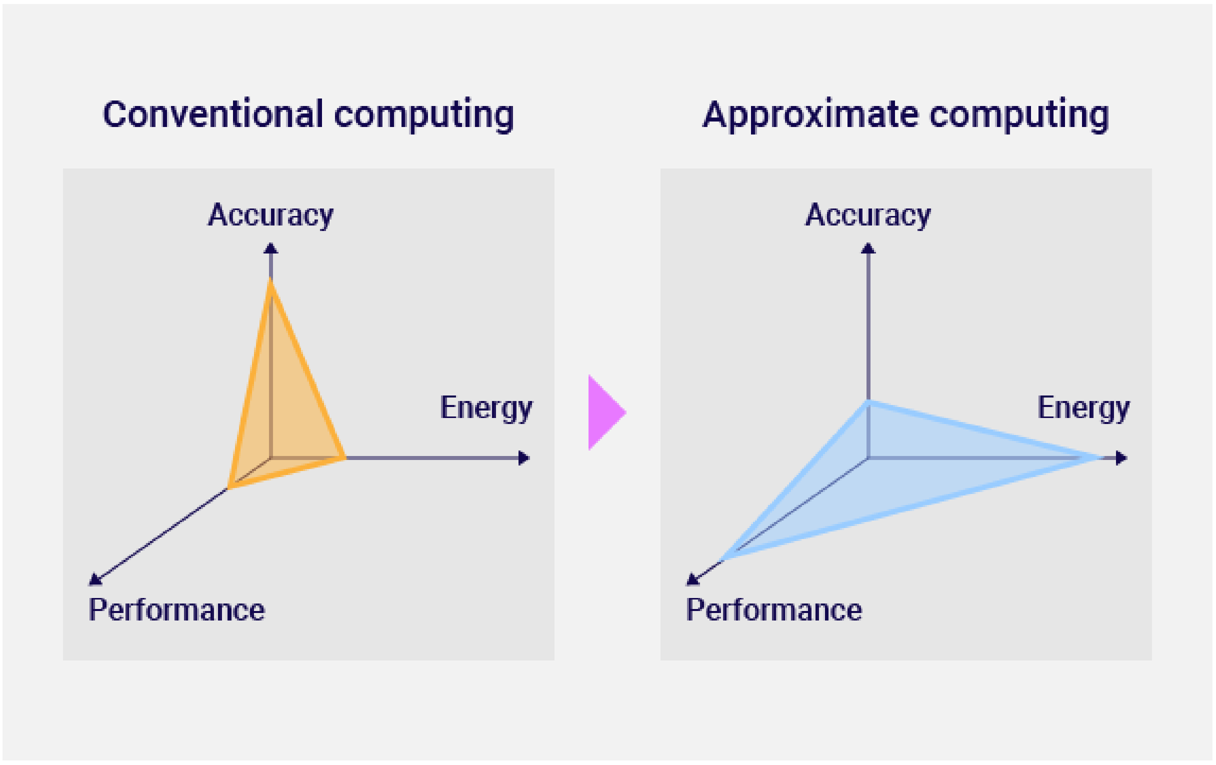


Figure 2.8: Approximated computing trade-off [17].

the application is able to tolerate. Approximated computing can instead take advantage of some statistical property of the data, in order to reduce the complexity.
The next chapters will consider approximated computing applied to molecular docking algorithms. This document does not provide a general framework that one can use to accelerate its application. This document wants to show how the in-depth analysis of the problem, can lead to a series of optimizations and approximations that increase the throughput of an application.

3 | State of the Art

This chapter will expose the most related work to this thesis, to provide an overview of the actual sate-of-art. A molecular docking algorithm is used to predict how a ligand binds with a receptor. Since the problem to approach is to speed up a virtual screening pipeline, we investigate how to apply approximated computing to the problem, by introducing a filtering step between the sampling and the scoring stage of the pipeline Figure 2.6. The purpose of the filtering stage is to filter out a part of the sampling's output, to reduce the input passed to the scoring stage. The sampling and scoring stages implementations are instead selected from a range of applications already present on the shelf. This chapter will review more in-depth the sampling and scoring algorithms in the literature, along with other tentative applications of approximated computing to virtual screening. This will give a better understanding of the effort driving this work.

3.1. Virtual Screening Pipeline

Compounds screening aims at finding the ligand with the highest interaction with the target protein. Among the various screening strategies of Listing 2.3 in this thesis, the focus is on high throughput virtual screening techniques. Figure 3.1 shows how the pipeline works. Starting from the pipeline inputs, we need the ligand structure and the receptor structure. A docking algorithm parses their structure, and by using a molecular docking

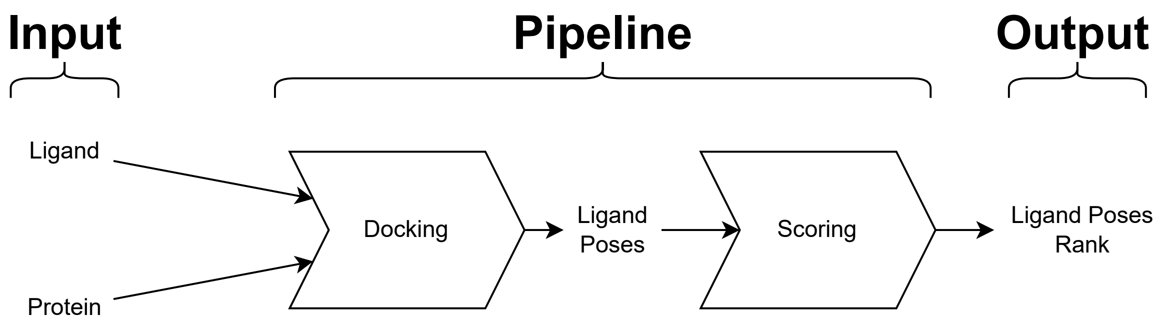


Figure 3.1: A virtual screening Pipeline representation.

algorithm, can return a set of ligand's poses inside the receptor's pockets. At this point, the scoring function ranks these ligand's conformations and extracts the lead compound conformations, that will be forwarded to later stages of the pipeline.
The virtual screening pipeline's success relies on the goodness of the scoring function. The higher the capacity of the scoring function of evaluating the binding affinity based on the displacement found from the docking stage. And it also depends on the sampling algorithm's capacity into estimating the 3D displacement of atoms upon interactions. Combining the two, we get that the higher the bond strength between a ligand's conformation and the receptor, the more likely the lead compounds will get good results in in vitro studies.

3.1.1. Docking algorithms

The previous chapter already mentioned different types of docking algorithms Listing 2.4. The process of docking a ligand onto a receptor can be approached in different ways: a way to take into analysis all the interactions between receptor and ligand, starting from an infinite distance among them, you move them closer and closer and simulate the full and real process of docking. This process is too computationally heavy to be carried out on a big dataset of ligands. The other way we consider is to immediately dock the ligand onto the receptor and evaluate the goodness of this docking. A classification of these algorithms is based on the degrees of freedom associated with receptor and ligand structures [18].

Rigid Body Docking Rigid body docking considers both ligand and receptor as a rigid structure, so only translations and rotations of the ligand are considered upon docking evaluation [18]. Rigid body algorithms utilize Fast Fourier Transform for energy evaluation [19]. Energy functions consider a small number of effects, like electrostatic and desolvation effects. The receptor's and ligand's structures are considered to be rigid. Examples of them are FRODOCK [20] and ZDOCK [21].

Semi-flexible Body Docking In Semi-flexible docking algorithms, the ligand is considered to be flexible in conformation, while the receptor is viewed as a rigid structure [18]. An example of semi-flexible body docking is LiGen [22].

Flexible Body Docking Orientation of the ligand inside the binding site alters the receptor structure. Thus considering either the ligand or the receptor as a rigid structure is no longer sufficient. This document focuses its attention on flexible docking algorithms since they are more accurate, but also more computationally expensive. Also in the

literature more examples of flexible docking algorithms can be found, each implementing different strategies. There exists Dock [23], Autodock [24], and many more [18].
Among them, we have modified the LiGen pipeline, by adding our filtering stage to it. LiGen has been our choice from the beginning of our work, also because it has gained attention after its usage in the Exscalate4Cov project [2]. E4C aims at using the EU's computing resources, to respond much faster to international pandemics. At the core of the project is EXSCALATE. EXSCALATE leverages a "chemical library" of 500 billion molecules, thanks to a processing capacity of more than 3 million molecules per second. Advanced CADD, in combination with the high throughput biochemical and phenotypic screening, will allow the rapid evaluation of the simulation results and the reduction of time for the discovery of new drugs.

3.1.2. Scoring algorithms

Docking algorithms produce as output a set of ligand poses. This set of poses can be very large based on the configuration of the docking simulation. The definition of the lead compounds requires the identification of the best-obtained pose. Poses have then to be evaluated, and ranked based on the evaluation. The evaluation given to the poses is called scoring, and the virtual screening component designed to do so is the scoring function. Scoring functions score ligand's conformations upon binding to the receptor based on different factors. The higher the score, the stronger the bond between the ligand-receptor pair will be, according to the used scoring function. Different scoring functions implementations exist, based on how they estimate the strength of interactions [25]:

- *Force Field Scoring Functions* (also referred to in the literature as physics-based [26]) considers physical interactions between atoms, including van der Waals effects, electrostatic interactions, and bond stretching between ligand and receptor. DOCK and Autodock scoring functions are an example of force field scoring functions. They are good at predicting the binding energy between ligand and receptor, but they are computationally expensive;

- *Knowledge-based Scoring Functions* use statistical-potential energy functions, derived from experimentally analyzed ligand-receptor binding. DrugScore [27] is an example of knowledge-based scoring functions. These kinds of scoring functions try to find a good balance between accuracy and speed. Fits of the energy equation are done using atoms structures, rather than reproducing atom affinities;

- *Machine-Learning-Based Scoring Functions* machine learning algorithms are used to evaluate a ligand's conformation upon docking, as the name suggests. The obtained results rely heavily on the training dataset. Machine learning scoring functions are not usually embedded inside scoring algorithms, instead, they are used for rescoring. Rescoring is used to confirm the results obtained with other scoring functions [10];

- *Empirical Scoring Functions* evaluate the binding affinity among ligand and receptor based on a set of weighted energy terms. These energy terms usually include van der Waals effects, hydrogen bond, rotatable bond, and hydrophobic effects. ChemScore [28], SCORE [29], and XSCORE [30] are examples of empirical scoring functions. Score computation is done with individual and intuitive additive terms, which yields a simpler calculation, and then a lower computational complexity compared with force field scoring functions, and they are good at predicting binding affinity [10].

Since our objective is to increase the throughput of a virtual screening scenario, empirical scoring functions are the ones we have taken into consideration. They are fast and produce a relatively good prediction of the binding affinities. In particular, XSCORE is the scoring function we have chosen for our application:

- LiGen scoring is inspired by XSCORE. So it is more useful for our case study, to have a pre-filter based on XSCORE, rather than with other scoring functions;
- XSCORE is open source, so the source code is easily accessible;
- XSCORE makes use of consensus scoring, the score associated with a pose is computed by interpolating multiple scores obtained by different scoring functions.

XSCORE

XSCORE is an empirical scoring function. XSCORE assumes that the binding score can be decomposed into basic components [30]:

$$\Delta G_{Bind} = \Delta G_{Motion} + \Delta G_{Interaction} + \Delta G_{Desolvation} + \Delta G_{Configuration} \tag{3.1}$$

Such terms are combined altogether, and their combination is calibrated starting from a training set of protein-ligand pairs, in a multivariate regression analysis.
XSCORE makes use of consensus scoring[31]: in this way it can ensure convergence results in the prediction, by having a final score that resembles the contribution of more factors.

The three functions used in consensus scoring are the following:

$$HMSCORE = C_{vdw,HM} \cdot VDW + C_{HB,HM} \cdot HB + C_{HM} \cdot HM + C_{RT,HM} \cdot RT + B_{HM} \tag{3.2a}$$

$$HPSCORE = C_{vdw,HP} \cdot VDW + C_{HB,HP} \cdot HB + C_{HP} \cdot HP + C_{RT,HP} \cdot RT + B_{HP} \tag{3.2b}$$

$$HSSCORE = C_{vdw,HS} \cdot VDW + C_{HB,HS} \cdot HB + C_{HS} \cdot HS + C_{RT,HS} \cdot RT + B_{HS} \tag{3.2c}$$

VDW is the contribution to the score associated with Van der Walls Interactions, HB instead refers to the Hydrogen Bonding Interactions, RT accounts for the rotatable bond effect. HMSCORE, HPSCORE, and HSSCORE differ due to the regression coefficients, and on the different effects they consider:

HM is the Hydrogen Matching contribution to the score. It was also adopted by SCORE [29] scoring function;

HP is the Hydrogen Contact Algorithm contribution to the score. It is inherited from the Chem-Score [28] scoring function;

HS is the Hydrophobic Surface contribution to the score.

Each term is associated with a Regression coefficient $C_{x,y}$, which is the regression coefficient associated with the X factor (HB, VDW, HM,etc.) inside the Y function (HSSCORE, HMSCORE, HPSCORE), plus the regression constant B_Y, one for each function. Finally all these ingredients are mixed altogether with a linear interpolation:

$$XSCORE = \frac{(HMSCORE + HPSCORE + HSSCORE)}{3} \tag{3.3}$$

This last computation introduces consensus scoring.

Autodock

From the Autodock algorithm [24] we have considered a specific component of its scoring function. Both XSCORE and Autodock's scoring function account for the hydrogen bond effect on the bond strength:

$$Autodock'sScore = VDW + HB + ELECT + CONFORM + TOR + SOL \tag{3.4}$$

Each term captures the contribution to the final scoring of Van der Walls Interactions, hydrogen bonding interactions, electrostatics effects, conformation effects caused by deviations from covalent geometry, torsion effects due to the rotation, and desolvation effects upon binding. Later on in the thesis, at Section 4.1.3 a closer look at the computation of the HB component is done [32].

3.2. Approximated Computing

Approximate computing increases the efficiency of the machine and technology already utilized, by exploiting the performance-quality trade-off. Approximate computing is a computing technique that relies on the capabilities of the application to be tolerant of the loss of accuracy in the results [33]. It can be applied to different fields, from multimedia processing to machine learning, to scientific fields. In the scientific field, approximations are largely applied. In fact, to capture the complexity of the real world (for example when analyzing the forces between atoms), we only consider the most important factors, to maintain the computation feasibly and fast. In these fields, a single solution to a problem may not exists, but approximated solutions are acceptable as an answer to the problem.
No universally known approximated computing technique exists that well suits all applications [34]. As such when approaching a new application to approximate, we should rely on a deep understanding of the problem, the algorithm, and finally the code. An example of a high-level framework that can be used to implement a self-aware approximated computing application is mARGOt [35]. mARGOt introduces an adaptation layer to provide self-optimization capabilities. In the following of this thesis, the focus is instead on how domain-expert knowledge can be used to make approximations and increase the throughput of an application.q
The initial step to do when applying approximated computing to an application consists in identifying variables and operations that can be approximated. In this way, we can gain more insights into the problem characteristics. The next step consists in deciding which approximation strategy applies to these variables/operations. In the following we make some examples [34]:

- *Precision Scaling* which reduces the precision of the input to minimize memory and computation requirements [34];
- *Loop Perforation* consists in skipping some iterations of a loop [34];
- *Skipping Tasks and Memory Accesses* in this case the application selectively skips some tasks or memory access [34];
- *Voltage Scaling* to reduce the energy consumption of memory, increasing in the

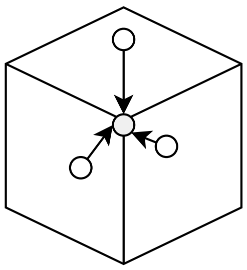


Figure 3.2: Continuous point inside the cube are represented by a single point among them, usually the center of the cube

meanwhile the probability of a bit flip during a read operation [34];

- *Neural Network accelerator* is used to emulate a code region, which can be approximated [34];
- *Memoization* approach works by saving the output of functions, that will be later reused when the same function is called on a similar input [34].

In this thesis, precision scaling and memoization are applied to the molecular docking pipeline. Precision scaling reduces the size of the inputs space. Now that the input space is finite, we can apply memoization, and so a pre-computation task can be carried out for all the possible inputs, and the results saved into memory. All the burden of calculus is moved away from the runtime application, to the pre-computation step. A similar idea has been proposed by qLUT [36]. qLUT suggests a general design paradigm, that in our case will be concretized for molecular docking. Quantizing the input introduces some approximations since now a set of input points are represented by a single one of them. Tuning approximation parameters, like grid spacing, is a critical point of approximated computing.

AutoGrid

Something similar to our approach is AutoGrid. Autodock, before running, requires the application's input to be analyzed by AutoGrid [37]. AutoGrid requires as input the target of our screening. From the receptor target of the analysis, AutoGrid builds a structure of pre-computed data. Summing up, AutoGrid uses precision scaling and memoization for the Autodock scoring function. Figure 3.3 shows how the pre-computed data are obtained from the 3D representation in space of the target receptor. The receptor's bounding box is gridified into points evenly distanced,. For each point Autogrid analyzes which is the impact on the final score if a ligand's pose has an atom in that position of a certain atomic type.

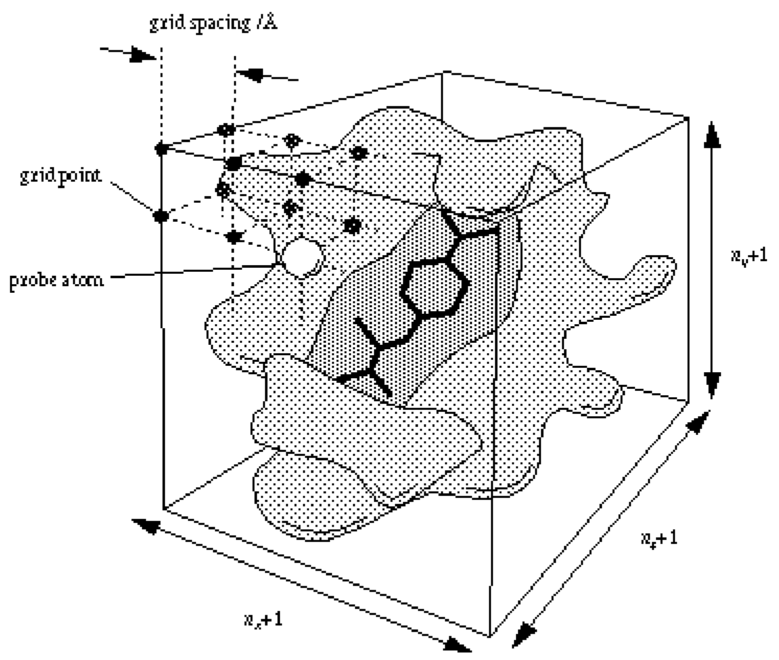


Figure 3.3: AutoGrid architecture [37].

Autogrid is taken as inspiration for the proposed methodology refined in this thesis. Because Autogrid's implementation is suitable to be used accordingly with Autodock, cannot be extended to other virtual screening pipelines or other scoring functions. This is different from the aims of this thesis, which wants to propose a methodology that can be used to apply approximated programming to different virtual screening pipelines. In Section 4.1 and Section 5.6.2 we will see how the proposed methodology of this thesis can be fitted on XSCORE and on the LiGen virtual screening pipeline.

3.3. HPC Infrastructure

High-performance computing (HPC) centers provide supercomputer that aims at maximizing the computation throughput. Supercomputers are usually organized as a cluster of nodes. To take full advantage of this infrastructure one should utilize all the nodes available. Today's top supercomputer [12] nodes are shipped with accelerators, which are usually GPUs. Full usage of a supercomputer can be obtained by working at different levels when implementing an application:

- At a design level, such that data and code are already targeting multiple nodes of the architecture. In this case, all the architecture characteristics can be utilized, but the code portability is reduced;

- At a programming level, by inserting a special directive inside the code, which will be later on be expanded by the compiler, when compiling for a specific architecture. In this way, the code portability is increased.

This work will make use of the second approach. We need to define which are the technologies stack we will use to balance the work among nodes.

3.3.1. Node Communication

The communication among nodes in a supercomputer is usually implemented by using Message Passage Interface [38], MPI is a standardization of a communication protocol architecture for parallel computing architecture. Since it is the de facto standard in HPC infrastructure. MPI is used to coordinate and synchronize the work of independent nodes with no concept of shared memory. It best suits our application since we want to develop it for an HPC infrastructure, with a very large number of nodes and we need to synchronize and balance the work among them. MPI adoption allows our solution to be highly scalable. The chosen MPI implementation is OpenMPI [39] since it is open-source, targets the GCC compiler, and is largely adopted. However other implementations of MPI exist, such as MPICH [40]. This thesis will adopt the usage of OpenMPI since we are more familiar with its usage.

3.3.2. Accelerator management

Since most of the top supercomputers nowadays utilize GPUs [12] we are interested in how we can take advantage of them to speed up our computation. At a programming level, we can use a different technological stack, like OpenACC, OpenGL, or CUDA. The choice among them is usually based on the vendor of the accelerator present on the targeting infrastructure.

OpenACC Open ACCelerators is a standardization of a parallel programming model targeting CPU/GPU architectures. It is integrated into a C/C++ application by using compile directives and code annotations. It works with Nvidia, AMD, and Intel accelerators.

OpenCL Open Computing Level is a framework for writing parallel workflow, which works on heterogeneous architectures, with CPUs, GPUs, DSPs, FPGAs, and other accelerators. It specifies a set of APIs that can be integrated into different applications, and also OpenCL specifies programming languages. It targets a large set of accelerators vendor: Intel, Xilinx, Nvidia, IBM, AMD, Qualcomm, and others.

CUDA Compute Unified Device Architecture is a parallel programming paradigm, which defines a set of APIs, that allow easy-to-use access to computational power exposed by Nvidia's GPUs. Our reference HPC infrastructure uses Nvidia's GPUs on each node, so the adoption of CUDA stack technology is mandatory.

SYCL Is a programming model, for programming hardware accelerators. Is a cross-platform abstraction layer, which gives the possibility of writing a single standard C++ code, which can target heterogeneous architectures. Since it works at a higher level, it does not requires the usage of any specific compiler: no C++ extensions are needed, making debugging easier. Among all these stack technologies, this thesis has adopted the usage of CUDA. Because the reference supercomputers infrastructure we have in mind utilizes Nvidia's GPUs as accelerators for each core. This choice is also confirmed by the real world's most adopted accelerator in supercomputers: the November 2021 top 10 supercomputers, which included 9/10 supercomputer architecture with an Nvidia accelerator per node. Thus the usage of CUDA is mandatory

4 | Proposed Methodology

This chapter explains the problem this thesis tries to target, and how we have used approximated computing to solve it. In particular, the proposed methodology used consists of a deep analysis of the application we are studying: the virtual screening pipeline. The analysis is required to identify approximations opportunities. Once the approximations opportunities have been located, it's time to understand which approximations strategies should be applied to them.
The structure of this chapter is organized into four sections. The first Section 4.1 illustrates the problem we are facing. Once the problem is well defined, Section 4.1 also defines the requirements for the tool resolving the problem, considering it as a black box. The application under analysis in this thesis is a virtual screening pipeline, so the next step is presented in Section 4.2, which analyzes how we have specifically fitted our tool to the virtual screening scenario. Finally, Section 4.2 introduces the architectural structure of the tool implementation, and how it has been adapted to XSCORE.

4.1. The Pose Filter

The problem we are facing in this thesis consists in accelerating the drug discovery process. Since the in vitro analysis of ligand and receptor requires time and money, it is common to use a computer-generated model to get a hint on which are the most promising inhibitors (ligand) of a given receptor. These molecules will be passed to later stages of the pipeline, for example for a more detailed in vitro analysis of the inhibitor. Inhibitor identification can be carried out using molecular docking in a virtual screening pipeline. Molecular docking algorithms generate ligand conformations that bind to the receptor, and then scoring functions are used to identify the most promising poses. Our goal is to increase as much as possible the throughput of a virtual screening pipeline. We anticipate some work done in the drug discovery process: instead of starting directly with in vitro testing, we start with a virtual screening pipeline to get the most promising compounds for further testing. The computation is done in silicon, in this way since we are using a computer-generated model the number of input poses is almost infinite. The limit on the number of

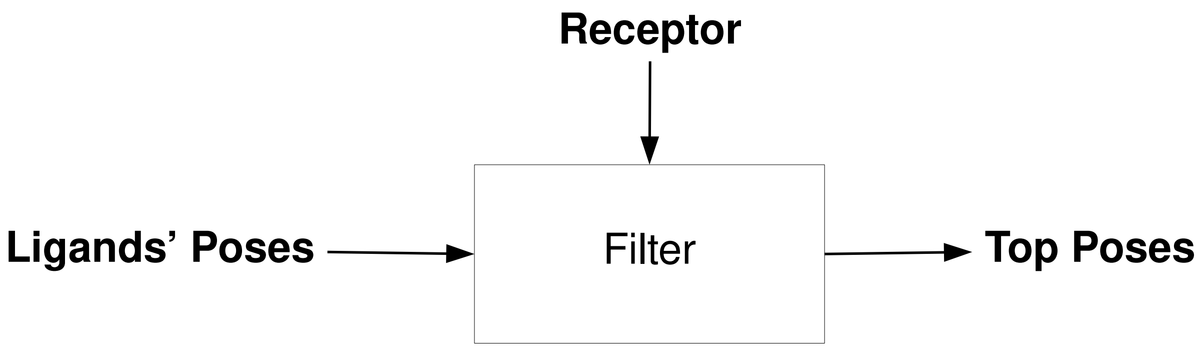


Figure 4.1: Filter block usage.

poses that can be scanned in a reasonable amount of time is not in the human and chemical resources available but the throughput of the application. Since the thesis' objective is to increase the throughput this means increasing the number of molecules analyzed, thus increasing the chance of finding better lead compounds. The results are then used as a hint for later stages of the pipeline, and they will be further verified by domain experts. The solution we propose is to use approximated computing on the scoring function, to speed up the computation.

The solution to the problem I propose in this thesis is the usage of a Filter. The Filter is a computational block, which is used to filter out the poses which are not likely to have a good binding affinity with the target of the analysis: the receptor. This can also be rephrased considering that an ideal scoring function should give a lower score to the poses with a low binding affinity with the receptor, so the Filter has to filter our poses that are likely to get a low score. This means that the Filter has to mimic the functionality of a scoring function. The Figure 4.1 shows the inputs and outputs of our block: as input, it requires the receptor under analysis in the virtual screening pipeline and the poses of the ligands produced by the previous docking step. As output, the Filter produces the best poses among the ones received as input. The computation done inside of the filter emulates the computation done by the targeted scoring functions. The filter approximates the scoring function. The filter can then be applied in two different scenarios. On one hand, it can be used as a replacement for a scoring function. On the other hand, it can be used as a filtering stage before the scoring stage of the pipeline. It can be helpful because it trade-offs computational complexity with accuracy on the final score. Or the Filter can be used as a filtering stage before the scoring function. In both scenarios, the remaining poses are passed to the real scoring function. The Filter has been developed by applying approximated computing technique to the virtual screening pipeline, using a two steps-process:

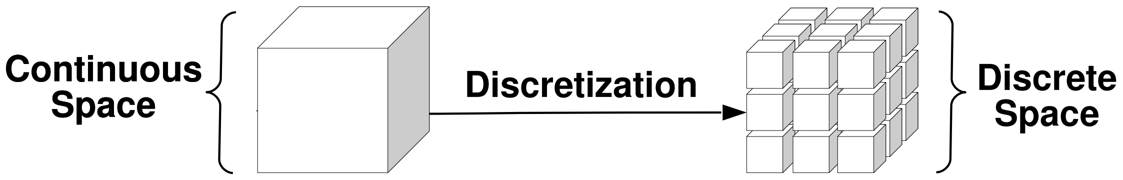


Figure 4.2: Input discretization process.

- the first step consists in the *identification* of the variables and operations which can be approximated. We can gain this knowledge by disassembling the application (XSCORE);
- the second step uses the approximation opportunities identified by the first step, and it decides which approximations *strategies* best target the approximation opportunities identified. This thesis makes use of precision scaling to quantize the input space, and memoization as shown in Figure 4.2

These two steps are better described in the next sections.

4.1.1. Approximations Identification

We will now disassemble how XSCORE computes each factor. This is the first step to gaining the required knowledge to identify the approximations opportunities present in the XSCORE implementation.

Van Der Walls Interactions (VDW)

As stated in the XSCORE paper [30] the VDW factor accounts for interactions between molecules that depend on distance. In particular, for each protein-ligand pair, we compute the VDW factor accordingly to the following equation:

$$VDW = \sum_{i}^{ligand} \sum_{j}^{protein} VDW_{ij} = \sum_{i}^{ligand} \sum_{j}^{protein} \left[\left(\frac{d_{ij,0}}{d_{ij}} \right)^8 - 2 \times \left(\frac{d_{ij,0}}{d_{ij}} \right)^4 \right] \tag{4.1}$$

Were $d_{ij,0} = r_i + r_j$ is the sum of the atom's radius, while d_{ij} is the distance between the two atoms. XSCORE implements this component with this Algorithm 4.1. This sketch of the algorithm shows how VDW factor computation is dependent form the atoms' radiuses and coordinates.

Algorithm 4.1 VDW interactions computation

```
1: for atoms in ligand that are valid and not hydrogen do
2:     for atoms in protein that are valid and not hydrogen do
3:         d0 = sum atoms radius
4:         d = distance between atoms
5:         if distance<cutoff then
6:             sum += Lennard-Jones 8-4 potential
7:         end if
8:     end for
9: end for
```

Hydrogen Bonding Interactions (HB)

HB factor accounts for the interaction that happens when two atoms get close enough and form a donor-acceptor pair [30]. As with VDW, also HB is computed pairwise for each ligand-protein atom couple:

$$HB = \sum_{i}^{ligand} \sum_{j}^{protein} HB_{ij} \tag{4.2}$$

Hydrogen Bonding Interactions are computed starting from the idea that a hydrogen bond has an ideal geometry, and any deviation from its makes weakens the strength of the interaction. So for each ligand-protein pair, the contribution to HB factor is computed as:

$$HB_{ij} = f\left(d_{ij}\right) f\left(\theta_{1,ij}\right) f\left(\theta_{2,ij}\right) \tag{4.3}$$

Where each term is computed as:

$$f\left(d\right) = \begin{cases} 1.0 & d \leq d_0 - 0.7\text{Å} \\ (1/0.7) \times (d_0 - d) & d_0 - 0.7\text{Å} < d \leq d_0 \\ 0.0 & dd > d_0 \end{cases} \tag{4.4a}$$

$$f\left(\theta_1\right) = \begin{cases} 1.0 & \theta_1 > 120 \\ (1/60) \times (\theta 1 - 60) & 120 > \theta_1 \geq 60 \\ 0.0 & \theta_1 < 60 \end{cases} \tag{4.4b}$$

$$f\left(\theta_2\right) = \begin{cases} 1.0 & \theta_2 > 120 \\ (1/60) \times (\theta 2 - 60) & 120 > \theta_2 \geq 60 \\ 0.0 & \theta_2 < 60 \end{cases} \tag{4.4c}$$

Also in this case $d_0 = r_i + r_j$ while d is the distance between the two atoms. Algorithm 4.2

Algorithm 4.2 Hydrogen bonding interactions computation.

```
1: for atoms in ligand that are valid do
2:     for atoms in protein that are valid do
3:         check which one is the donor and which one is the acceptor
4:         if cannot make hydrogen bond then
5:             skip this pair
6:         end if
7:         compute the distance between atoms
8:         if distance<cutoff then
9:             skip this pair
10:        end if
11:        score this hydrogen bond
12:        add this hydrogen bond to the candidates list
13:    end for
14: end for
15: for each candidate do
16:    for each candidate do
17:        check that the angular limit between two hydrogen bonds on the same atom is
           greater than 45°
18:        check that an acceptor atom cannot accept more hydrogen atoms than the number
           of lone pairs
19:        check that a donor atom cannot form more hydrogen bonds than the number of
           hydrogen atoms it has
20:    end for
21: end for
```

wraps up the implementation of the HB stepwise functions. By valid atoms, we consider ligands or protein atoms that are non-hydrogen, nitrogen, phosphor.
Moreover, HB computation requires atoms' radiuses and coordinates. We would like to stress out that HB computation depends also on the atom's root, defined as [30]:

> The root of an atom is its non-hydrogen neighboring atom. When an atom bonds with more than one non-hydrogen atom, its root locates at the geometric center of all its non-hydrogen neighboring atoms.

Hydrophobic Terms

XSCORE makes use of consensus scoring as detailed in this section 3.2. HPSCORE and HMSCORE account both for the hydrophobic effect. But the computation of the factor is done with two different approaches: Hydrophobic Contact and Hydrophobic Matching.

Hydrophobic Contact (HP) This algorithm was adopted by *Chem-Score*[28]. its value is computed by summing up the hydrophobic atom pairs between the ligand and the protein [30]:

$$HC = \sum_{i}^{ligand} \sum_{j}^{protein} f\left(d_{ij}\right) \tag{4.5}$$

With $f\left(d_{ij}\right)$ defined as:

$$f\left(d\right) = \begin{cases} 1.0 & d \leq d_0 + 0.5\text{Å} \\ (1/1.5) \times (d_0 + 2.0 - d) & d_0 + 0.5\text{Å} < d \leq d_0 + 2.0\text{Å} \\ 0.0 & d > d_0 + 2.0\text{Å} \end{cases} \tag{4.6}$$

Similarly to the previous factors of the computation, d_0, d are defined respectively as the sum of the radius, and as the distance between atoms. The score associated to a pair of atoms will have max value when the two hydrophobic atoms have a van der Wall interaction, and diminish as the distance increases. Algorithm 4.3 shows how the HP

Algorithm 4.3 Hydrogen Contact computation

```
1: for atoms in ligand that are valid do
2:     for atoms in protein that are valid do
3:         $d$ = distance between atom i and j
4:         if d>cutoff then
5:             skip this pair
6:         end if
7:         sum this contribution to atom's i total
8:     end for
9:     add total atom's i contribution to the score
10: end for
```

factor is computed in XSCORE. With valid atoms we refers to non-hydrogen atoms. Also in this case the Algorithm 4.3 shows data dependencies on atoms' radiuses and coordinates. To be noted that Line 7 compute the contribution of this pair in the same way as done by hydrophobic matching algorithm, at Line 7.

Hydrophobic Matching (HM) HM factor evaluations is done considering that different parts of the ligand sense the protein differently because of the heterogeneous nature of the binding site. If a hydrophobic ligand atom is placed at a hydrophobic site of the protein, then it is expected to be favorable to the binding process [30]. The computation of the HM component is taken from an algorithm used in SCORE [29]. HM values for a pose is computed as in the following:

$$HM = \sum_{i}^{ligand} \log P_i \times HM_i \tag{4.7}$$

With HM_i is an indicator variable:

$$HM_i = \begin{cases} 1 & \text{if hydrophobic atom in an hydrophobic environment} \\ 0 & \text{if otherwise} \end{cases} \tag{4.8}$$

$\log P_i$ is the contribution of atom i to the n-octanol/water partition coefficient of the molecule. This scale factor depends on the atom's type, defined as its atomic type, so they can be computed once and then saved [41]. $\log P_i$ is used as a weight factor, such that hydrophobic atoms contribute more to the hydrophobic effect.
This factor takes into account that the ligand reacts differently when docked into a different part of the docking site, due to differences that can be in the docking site nature: in fact, a hydrophobic ligand's atom is favored in a hydrophobic protein's site. HM, factor

Algorithm 4.4 Hydrogen Matching computation

```
1: for atoms in ligand that are valid do
2:    for atoms in protein that are valid do
3:       $d$ = distance between atom i and j
4:       if d>cutoff then
5:          skip this pair
6:       end if
7:       sum this contribution to atom's i total
8:    end for
9:    if $\log P_i \geq 0.5 \vee total_i > -0.5$ then
10:      score = $\log P_i$
11:   else
12:      score = 0.0
13:   end if
14: end for
```

si computed in XSCORE with Algorithm 4.4. Valid atoms refer to ligands and protein atoms that are not Hydrogen, and only ligand's atoms that have an $\log P_i$ that is greater than 0. HM is another factor for which the computation is dependent on the atoms' radiuses and coordinates. Is to be noted that Line 7 computes the contribution of this pair in the same way as done by the Hydrophobic Contact algorithm, at Line 7.

Hydrophobic Surface (HS)

HS factor is assumed to be proportional to the buried hydrophobic surface of the ligand [30]. This is the most computationally intensive part done during XSCORE computation. Before actually computing this factor value, XSCORE requires the definition of the solvent-accessible surface, which is represented by evenly distributed point spacing 0.5 Å on the ligand surface. It is computed using a probe atom with a radius of 1.5 Å, which is considered to be buried if it penetrates the SAS (solvent-accessible surface is the area of a molecule accessible by a solvent) of the protein:

$$HS = \sum_{i}^{ligand} SAS_i \tag{4.9}$$

So for each ligand, we are analyzing we should compute the SAS with Algorithm 4.5.

Algorithm 4.5 Surface Dots generation algorithm

```
1: for ligand atom that is valid do
2:     create a sphere centered in this atom with a radius sum of the atom's radius and
       the probe's radius
3:     generate a candidate list of evenly distributed point on the previous sphere
4:     for each candidate in the list do
5:         if is on the ligand surface then
6:             add the point to the surface's point list
7:             associated with its unit
8:         end if
9:     end for
10: end for
```

The point Unit is the portion of the surface represented by this point after discretizing the surface. In this case is defined as $\frac{\text{sphere's surface}}{\text{points on the surface}}$. Valid atoms are non-hydrogen atoms. In this algorithm, the surface is quantized into a finite number of points. Each point represents an area on the sphere's surface. The check if a candidate point is on the ligand's surface is carried out by comparing its distance with each other ligand's atom position k: if this such a distance is 10% more than any other sum of atom's k radius and the probe atom's radius, it is considered to be on the surface.

Once we have obtained the points that are both on the protein's and ligand's surface we can proceed with Algorithm 4.6 computation. Valid ligands' atoms are the ones that are non-hydrogen, phosphor, nitrogen, and the ones that are acceptors. Protein atoms are valid if they are not hydrogen atoms. Wrapping up each ligand's atom the algorithm checks the protein atoms for which is not possible to put two water molecules between them. Then it checks which of the surface point generated by the ligand's atom i have a

Algorithm 4.6 Hydrophobic Surface factor computation

```
1: for ligand atom i that is valid do
2:     for ligand surface point p generated by atom i do
3:         for protein atom that is valid do
4:             d = distance between point p and protein atom j
5:             if $d < r_j + WATER\_R$ then
6:                 sum up the point Unit of point p to atom i
7:             end if
8:         end for
9:     end for
10:    atom score is the ration between the atom buried surface and the total surface of
       the ligand
11: end for
```

distance that is less than a water probe from the protein atoms. The unit is associated with this point will contribute to the ratio, and then to the final impact on the score. The combination of the two algorithm highlight how also HS computation requires atoms' radiuses and coordinates. The problem is that HS computation also requires the definition of the ligand's SAS which cannot be pre-computed. This is a similar problem we have encountered for HB computation, and which is better analyzed in section 4.1.3.
Is worth noticing that this algorithm has been first used in Bohm's scoring function [42].

Deformation Effect (RT)

Upon binding, ligand and protein are constrained in conformation due to atom interactions, with respect to their solvent state. The deformation effect of the ligand is then expressed as the contribution of all the rotors with proper weight. This introduces a negative factor to the scoring value since it reduces the interaction strength. Rotatable bonds are important to understanding molecular flexibility [43]. RT is computed as: $RT = \sum_i^{ligand} RT_i$ Where RT_i is a step function:

$$RT_i = \begin{cases} 0.0 & \text{if atom i not involved in any rotor} \\ 0.5 & \text{if atom i is involved in one rotor} \\ 1.0 & \text{if atom i is involved in two rotors} \\ 0.5 & \text{if atom i is involved in more than two rotors} \end{cases} \tag{4.10}$$

RT value is computed with Algorithm 4.7. RT algorithm shows dependencies on the ligand structure, as for HB and HS its pre-computation is not straightforward. This problem is further analyzed in section 4.1.3.

Algorithm 4.7 Deformation Effect computation

```
1: for ligand bonds that are valid do
2:     mark non valid bond, line ring bonds, non-single bonds, terminal rotors
3: end for
4: for ligand atoms that are valid do
5:     count the number of bond not marked in which it is found
6:     compute the score of the atom based on the step function previously defined
7: end for
```

Apparent Charge

XSCORE assigns an apparent charge to each atom of the pose. This is done before each pose is scored. The apparent charges value depends on the atom type: these values are loaded from pre-computed values.

4.1.2. Approximation Strategies

Now that we have gained domain knowledge, we can apply approximations to the application. In particular, we will apply memoization and precision scaling to XSCORE. Memoization has been implemented by making extensive use of fast lookup tables.XSCORE has as input ligand atoms, and each XSCORE factor depends in general on three properties of the atoms:

- atoms radiuses;
- atoms types;
- atoms coordinates.

Coordinates and radiuses are real numbers, so an infinite combination of them exists. Precision scaling is applied at this point since the pre-computation needs to discretize the inputs, as in Figure 4.2. In this way, we obtain a finite number of input combinations. Up to now, the Filter has been considered as a black box, we now enter inside of it to analyze its component. Figure 4.7 shows how the Filer block emulates the scoring function. The inputs are the receptor structure and the ligand's poses. Internally the Filter works in two steps:

- a *Pre-computation* step uses the receptor structure for the memoization and precision scaling part. It has to be done once for each receptor. Precomputed data are passed on to the next step;
- *Filtering* step with the approximated version of the scoring function. The approxi-

mated scoring function will score ligands' poses based on the precomputed data of the previous step.

The output of the Filter will be a ranking of the input poses.

4.1.3. How to Approximate XSCORE

The following part of the thesis defines how the previously described techniques and the knowledge of XSCORE, can be merged to approximate XSCORE computation. For each component of the XSCORE function, we will analyze how it can fit in our Filter block, and in particular how we can anticipate its computation in the pre-computation stage.

Apparent Charge

Before each pose is scored, XSCORE re-assigns the apparent charges. In the end, XSCORE assigns a charge of 0.0 to all atoms except for atoms of type: 0.co2, N.4, and N.pl3.h. We can then approximate it to assign directly 0.0 to all atoms.

VDW, HM, HP

Van der Waals interactions, hydrophobic matching, and hydrophobic contact function computations show dependencies with ligands' atoms coordinates and radiuses. So it is possible to pre-compute grids associated with these functions, by using an atom probe that changes its radius for each point of the grid (as detailed in 5.1.2).

HB

HB factor computation in XSCORE shows dependencies with: the atom's coordinates, atom's radius, and type, and also with the atom's root. Atom's root computation requires knowing the atoms near him, that are in the same compound pose (ligand or receptor). At the pre-computation time, this is impossible for ligand's pose atoms. This is a problem since the HB factor appeared in all the three functions used by XSCORE in the voting system (HSSCORE, HMSCORE, HPSCORE). From the analysis in figure 4.3, we've found out that HB values account on average for 2% of the final score value. Since the HB factor contributing to the final score cannot be pre-computed using the XSCORE HB algorithm, we have made three attempts in approximating HB:

- *Full HB*: in this case, HB computation remains unchanged, the approximation on the HB factor is 0;
- *No HB*: in contrast with the previous approach HB value are set by default to 0;

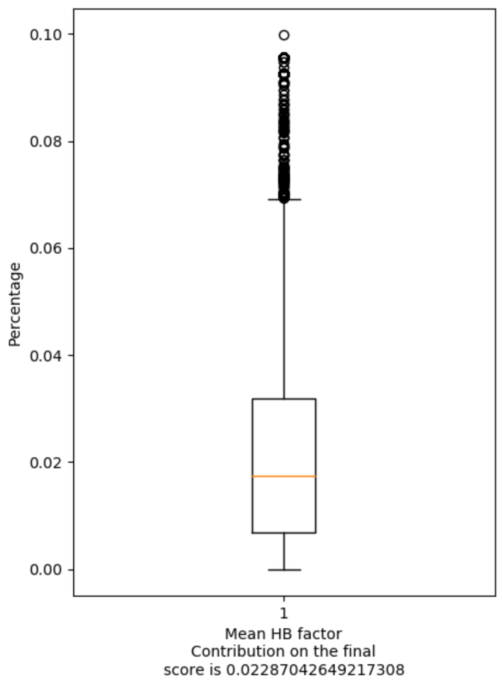


Figure 4.3: How much the value of HB assigned to each pose contribute to the final score in XSCORE.

- *HB Approximation*: since XSCORE HB algorithm cannot be precomputed due to the atom's root dependency. This dependency has an impact on the first in the HB equation 4.1.1. Since the three-term in equation 4.1.1 ranges from 0 to 1, we can try to approximate the equation by setting the term dependent on the angle between the hydrogen bond and the root of the atom always to 1:

$$HB_i = f(d)\, f(\theta_2) \tag{4.11}$$

  The HB algorithm in XSCORE has also other data dependencies: on the atom probe capacity of accepting or donating electrons (Donor, Acceptor, Donor-Acceptor), and on the atom probe radius. So for each grid point, we should store a different value for each combination of atom probe type and radius;

- *Autodock HB*: HB values are instead pre-computed using the HB algorithm of Autodock, which can be pre-computed. Autodock HB computation algorithm is described in paper [32]: $\Delta G_{AutoDock4} = \sum_{i,j} E(t)\, E(\theta) \left(\frac{C_{ij}}{r_{ij}^{12}} - \frac{D_{ij}}{r_{ij}^{10}} \right)$ Where $E(t)$ is a function of the angle of the probe atom away from the ideal position for hydrogen bonding. While $E(\theta)$ is a ramp function that favors hydrogen bonding interactions that occur with the trigonal or tetrahedral geometry of the lone pairs. In our case, we have found AutoGrid very useful for our purposes. AutoGrid is a tool used by Autodock for pre-computing required data structures that will speed up computation. These data structures also include the HB values pre computation for each point of the grid. The HB pre-computation algorithm used in AutoGrid has data dependencies on the atom type. The outputs of AutoGrid HB computation will be a set of files. Each file will contain the grid with the HB value in space of an atom probe, with atom type associated with the file name.

Above all these approaches we have chosen our final version for the thesis the version with Autodock HB. Due to the good approximation, we have obtained when using this version, as we will demonstrate in chapter 5.
The implementation details for HB computation using the Autodock algorithm are described in section 4.2.2.

RT,HS

RT computation is set by default to 0 since it is a property of the ligand which is independent of its pose. RT value is then the same for each pose, and thus its impact on the final value of the score will be the same for each pose.

HS instead cannot be pre-computed, its computation depends on the input ligand because the computation of the Solvent Accessible Surface (SAS) is needed. Since XSCORE is a consensus scoring function, the HS factor is less important because it is used only in the HSSCORE function, one of the three functions used for voting. In our XSCORE Filter version, we have disabled HS computation, due to the complexity of computing the SAS. Also from figure 4.4 we know that its impact on the score is quite small, only 4%. It is also less important as a factor because the HS factor compares only to the HSSCORE function, which is only one of the three functions for voting. This is different from another factor like HB, which is instead used in all three voting functions.

4.1.4. Filter Implementation

We now introduce how we have implemented the approximations proposed in Section 4.1.2, by splitting the Filter into two blocks: the Pre-computation and Filtering stages.

Pre-computation Stage

The pre-computation stage is required once for each receptor of a virtual screening analysis. The fast lookup tables previously mentioned, have to be calculated in advance before

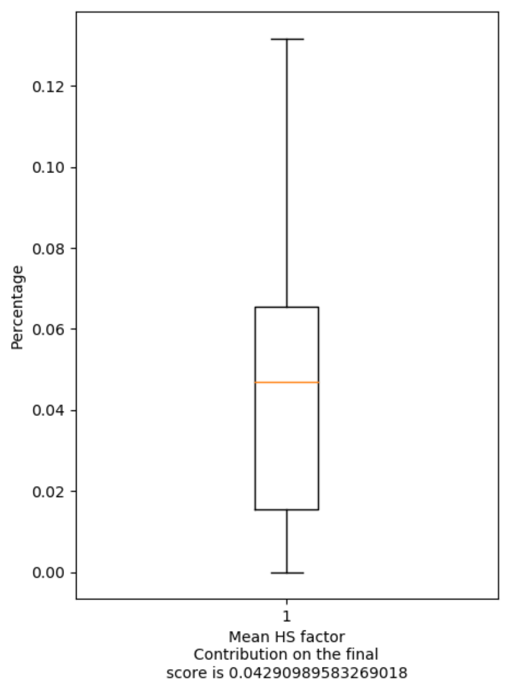


Figure 4.4: XSCORE HS values impact on the final score.

the real computation starts. At run time, they can be accessed in constant time, removing a lot of overhead. All the returned values, obtained from a function call with as input all the available input space, will be stored in these tables. Pre-computing all functions results requires exploring the whole input space. For each component (HB, VDW, etc.) these tables are organized as a grid, which should be pre-computed. Each XSCORE component computation depends on different factors (atoms radiuses, types, coordinates). Grids are a 3D matrix in which each cell represents a position of the space, in which we place an atom probe for evaluating a function return value. Each point of the grid stores multiple results of the same function, based on the atom's probe properties. Atom's properties can change, in our case atom probe can change radius, type, or both. Since radiuses are a real value, they are discretized along with the atom probe position in space.
Once a function has been precomputed it is saved in memory. At runtime, all grids are loaded into different lookup maps. Maps are then accessed in constant time. Grids and maps are what we have called lookup tables. The grid's structure is defined starting from the fact that atoms coordinates are required as input of each XSCORE component.
The grid's structure is defined by two factors:

- **Grid Spacing** this is the space between two consecutive points on the same axis

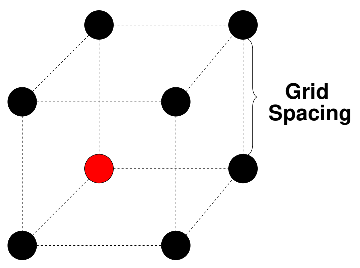


Figure 4.5: Grid spacing definition.

In Figure 4.5 represents a portion of the quantized space. The cube vertexes are the points of the grid, obtained after the space has been quantized. The red central point represents any of the points of the continuous space, which will be represented by the nearest vertex of the cube. The loss of accuracy from this approximation is because the real atom position is approximated to the near cube's vertex.

- **Protein's Dimensions** Atom probes are placed in each grid's position. The grid's dimension will be a bounding box surrounding the area of interest around the receptor.

Each grid point stores multiple values of the same component, based on the atom's properties on which the factor computation depends:

- *Atom Type* each function has a different grid, each grid point stores a value for each atom type. Atom type refers to its atomic type (nitrogen, hydrogen, etc.) or to its ability to accept, or donate an electron;
- *Atom Radius* each function has a different grid, each grid point stores a value for each atom radius;
- *Atom Radius and Type* each function has a different grid, each grid point stores a value for each combination of atom radius and type.

During the pre-computation stage for each point of the grid, we compute the contribution to all the factors (VDW, HB, HM, etc.) of an atom probe at that point. The atom probe is not fixed, it changes some of its properties (radius, type, etc.), so at each point of the grid, we store multiple values for each XSCORE factor, and each combination of atom probe properties.
The pre-computation of each factor of a scoring function is carried out by placing an atom probe in each point of the grid associated with that factor. For each position, the atom probe can change its properties, based on the computation dependencies of the factor related to that grid. For example, VDW computation requires to use an atom probe that

changes its radius in each position of the grid, to pre-compute the contribution to the score value, if an atom is placed in that position.

Atom Radius At each point of the grid, a different atom probe is placed. Atom probes can differ based on their radius. An atom probe placed in a certain grid point can change its atom radius. Radiuses have to be quantized since they are real values. The approach we have used consists of grouping atoms' radiuses into ranges. For each range, we define a *max* value, a *min* value, and an *average* value, which will be the radius value used during pre-computation for all the atoms that have a radius between max and min value

Atom Type The pre-computation of some factors like HB requires to use an atom probe which changes its atom type in each position of the grid. An atom probe placed at a certain grid point can change some of its properties. For example, for each available atom type in a certain point of the grid, the function under pre-computation is called, and the results are stored in the grid. With atom types, we refer to both the atomic type (hydrogen, nitrogen, etc.) and the atom's ability to accept or donate electrons (Donor, Acceptor, Donor-Acceptor). An increase in the number of atom types increases the memory requirements since more values per point of the grid should be stored because for each point of the grid we need to pre-compute the value for each combination of atom types and radiuses. If the atom type is considered to be the atomic type (nitrogen, hydrogen, etc.) then the radiuses are already known and no other layer has to be introduced based on the radiuses. Instead, if we consider the atom type to be Donor, Acceptor, or Donor-Acceptor we should compute other layers for each type based on the possible radiuses.

Pre-computation algorithm The algorithm used for pre-computation is very straightforward. It consists in moving an atom probe on the grid, and for each point of the grid, its contribution to the factor value (VDW, HB, etc.) is computed. As described in Algorithm 4.8. Whenever points are all points that have a distance from protein's atom j that is lower than the value of cutoff. We have customized this algorithm for each factor we are pre-computing. For example for VDW, HM, and HP factors we have implemented Algorithm 4.9 because they have data dependencies on atoms' radiuses and coordinates.

Another example is the pre-computation of the HB factor. In this case, we have implemented two algorithms: the first Algorithm 4.10 considers the HB pre-computation dependent on the atomic types (each type having its radius), while the second Algorithm 4.11 considers HB dependent on the atoms' capacity to donate or accept electrons, and on the atom's radiuses.

Algorithm 4.8 Factor pre-computation algorithm, which will be structured as a grid.

```
1: for valid protein atom j do
2:     for point near protein atom p do
3:         factor kernel pre-computation based on factor data dependencies
4:     end for
5: end for
```

Algorithm 4.9 Single grid point pre-computation algorithm, for factor with data dependencies on atoms' radiuses and coordinates.

```
   Input receptor atom j, grid g, grid point p
1: for radius ranges with average radius r do
2:     compute the contribution to the factor of grid g, due to the relation between receptor's atom j and atom probe with radius r in position p on-grid g.
3: end for
```

Algorithm 4.10 Single grid point pre-computation algorithm, for factor with data dependencies on atoms' atomic types.

```
   Input receptor atom j, grid g, grid point p
1: for atomic's types t do
2:     compute the contribution to the factor of grid g, due to the relation between receptor's atom j and atom probe of atomic's type t in position p on-grid g.
3: end for
```

Algorithm 4.11 Single grid point pre-computation algorithm, for factor with data dependencies on atoms' capacity of accepting or donating electrons.

```
   Input receptor atom j, grid g, grid point p
1: for atom's type in [Donor, Acceptor, Donor-Acceptor] as t do
2:     for radius ranges with average radius r do
3:         compute the contribution to the factor of grid g, due to the relation between receptor's atom j and atom probe of atom's type t with radius r in position p on-grid g.
4:     end for
5: end for
```

Pre-filtering stage

The Filtering stage uses as input the data coming from the pre-computed stage to evaluate the ligand's poses coming from the docking algorithm. The filtering stage implements the approximated version of XSCORE. Algorithm 4.12 further explains the importance of the

Algorithm 4.12 Filtering stage algorithm implementation.

```
   Input output grids of pre-computing stage G, ligand's poses
   Ouput ligand's poses ranking
1: for ligand's pose p do
2:    for pose's p atom a do
3:       approximate atom a coordinates to a point on the grid
4:       select grid g from G based on atom a properties
5:       uses g data to evaluate the contribution to each factor of the scoring function of
         atom a.
6:    end for
7: end for
8: compute the rank
```

pre-computation stage to increase the throughput: the contribution of each atom to the factors of the scoring function has been already computed in the pre-computation stage. So when Line 5 is reached no computation has to be done. As the last step computes the poses' ranking based on the previous evaluations.

4.2. Solution Architecture and Usage

Now the focus is shifted to the implementation of the Filter for a virtual screening pipeline and how it can be used. We want to propose a general approach that can be used to develop a Filter based on the application. In this thesis, the application is a virtual screening pipeline. The original virtual screening pipeline is presented into Figure 3.1. The output of the docking stage is fed directly to the input of the scoring stage. In the modified pipeline proposed by this thesis, in Figure 4.6, the Filter stage can be inserted inside a virtual screening pipeline, in between the docking and scoring stage. The Filter takes as input the ligands' poses obtained from the docking stage and produces as output a subset of poses that has to be processed by the scoring stage. Using the Filter in this setup, allows the modification to be transparent to the docking and scoring stage. Moreover, the docking stage can produce as output more poses to be evaluated since the Filter can select only the most promising ones. The best poses are then passed to the scoring stage.

Since in a virtual screening pipeline a protein is usually docked with a lot of ligands, it is

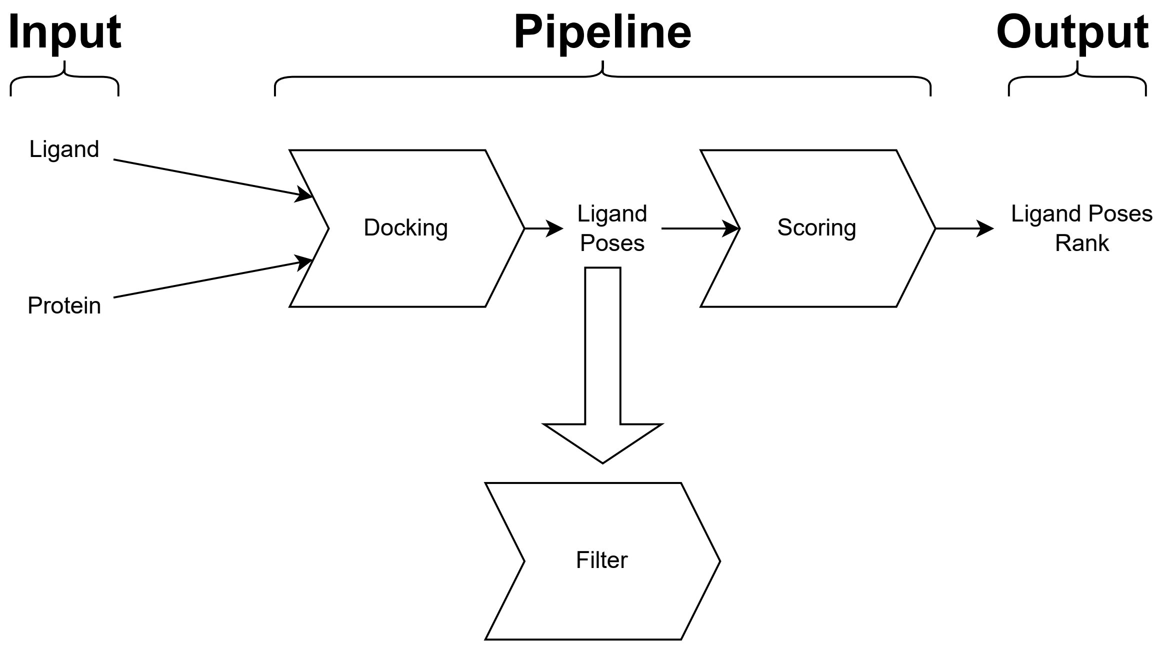


Figure 4.6: High throughput poses selection with pre filtering

convenient to split the Filter block into two smaller components to be used in a different part of the pipeline. Figure 4.7 illustrate how the filter can be used from an external point of view, as a black box, which receives the receptor's and ligands' structures and gives as results the top poses. A closer look at the filter is found in Figure 4.8, which shows how the Filter is decomposed into:

- The Pre-computation stage that has to apply memoization and precision scaling using the receptor's structure;
- The Filtering stage will score with the approximated version of the scoring function the input ligand's poses and the precomputed data from the previous stage. This stage has the job of filtering out the poses which more likely will get a low score.

The Filter's output will be a ranking of the poses produced by the docking stage. Considering the Filter as composed of two smaller building blocks has the advantage of making the Filter re-usable and more efficient. For example in an HPC environment, the workload is split into batches. So for each receptor, the ligands' poses to be analyzed are grouped into batches. Each batch is processed separately from the other: the virtual screening pipeline is run into different instances, maybe in parallel. The pre-computation stage is the same among all pipeline runs because the receptor remains the same. Is then highly inefficient to perform pre-computation at each pipeline run. Is more clever to do the pre-computation once, and store the result in memory. Then every launched pipeline

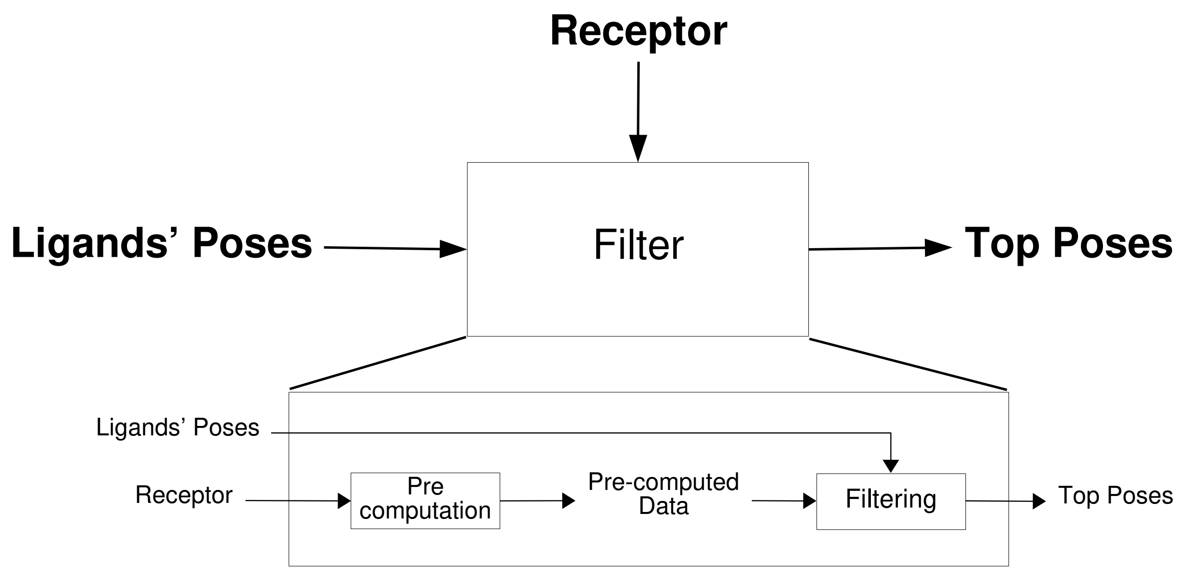


Figure 4.7: Filter block disassemble. This figure shows how the Filter can be composed of smaller building blocks: the pre-computation and filtering blocks.

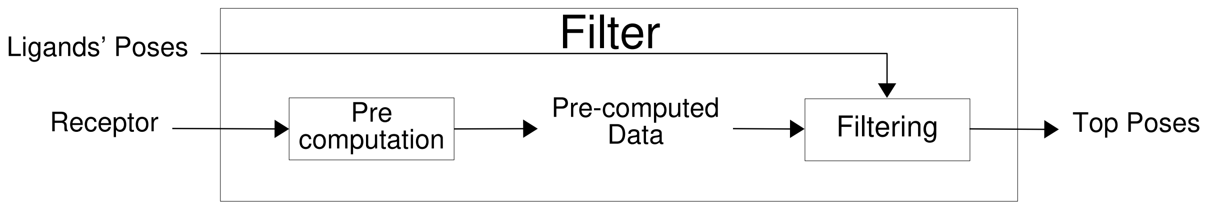


Figure 4.8: Filter block close look to its internal structure.

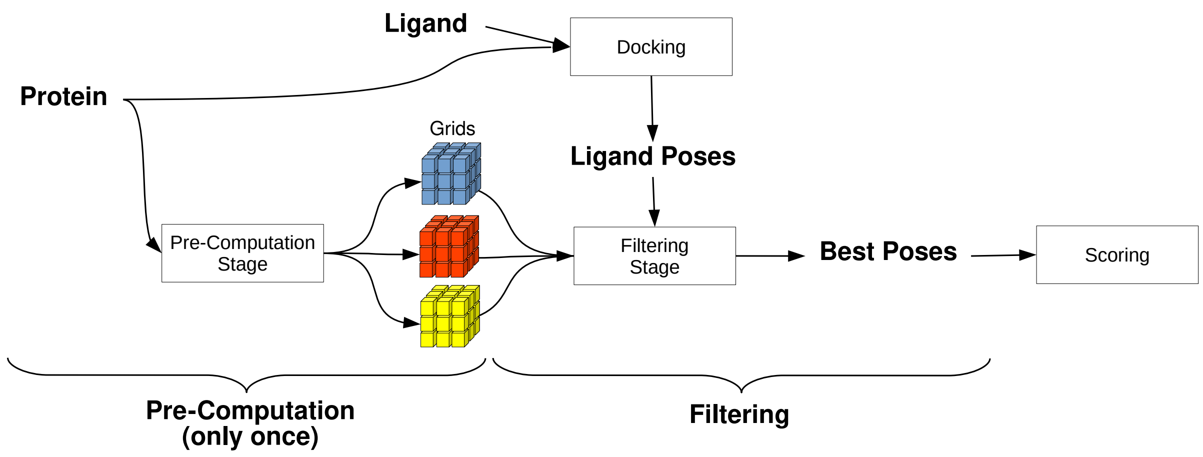


Figure 4.9: Filter insertion into a virtual screening pipeline.

will load into memory the pre-computation. The pre-computation result is organized into memory in multiple grids, as we will see in the next sections. Wrapping up, our complete pipeline is described in Figure 4.9. Before launching the virtual screening pipeline, the pre-computation stage takes the protein's structure and computes the grids required by the filtering stage. The filtering stage takes the ligand's poses produced by the docking algorithm, and it evaluates them using an approximated scoring function and the pre-computed data coming from the pre-computation stage. This architecture has also the advantage that the docking and scoring algorithms are unaware of the filter's existence, they remain unchanged.

4.2.1. CUDA & MPI Implementation Architecture

To get the best speedup we should also go beyond the optimization of the scoring algorithm: we should take advantage of the underlying technology that will run our application. The next section will review the results we have obtained with the last version of our application, that fully embrace the usage of MPI and CUDA technologies. So our final implementation will be available in C/C++, because XSCORE implementation is written in C/C++, and also CUDA and OpenMPI are targeting C/C++.
The complexity of the Molecular Docking problem requires it to run on a supercomputer, and so scalability is needed since it will target multiple nodes. The best way to cope with this is the usage of MPI, the de facto standard when doing HPC on a supercomputer. With our MPI implementations, we want to address the problem of scoring a very large dataset of ligands against a single target receptor. To better understand this, figure 4.10 shows how computation is balanced among nodes. After each node produces its rank, we collect all ranks, and we merge them into only one ranking. From this final rank, we can

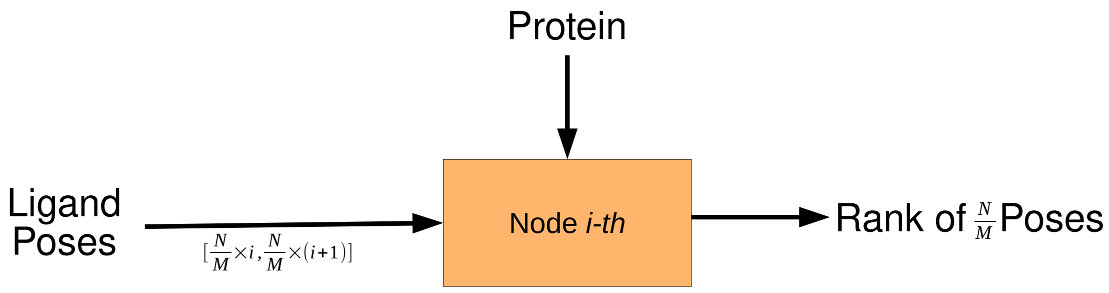

Protein
Ligand Poses
$[\frac{N}{M}\times i, \frac{N}{M}\times(i+1)]$
Node *i-th*
Rank of $\frac{N}{M}$ Poses

Figure 4.10: MPI architecture

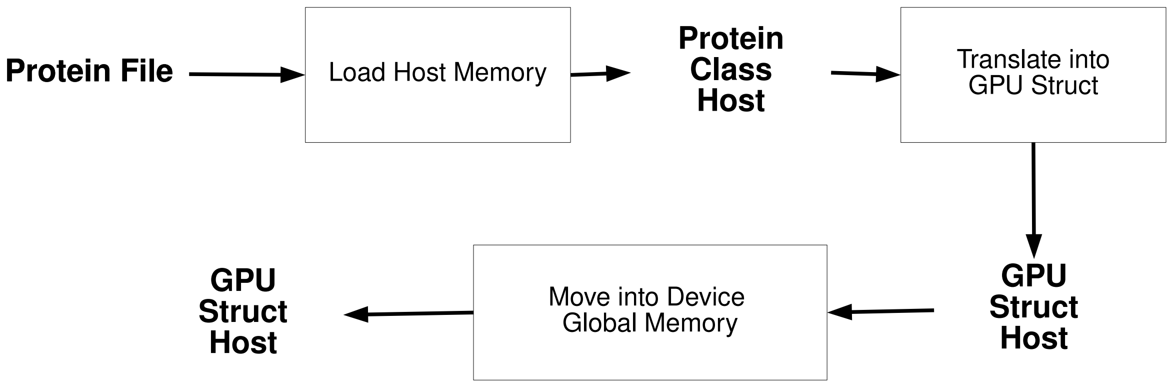


Figure 4.11: GPU memory loading architecture.

retrieve the best poses that should be passed as input to the next stage in the pipeline. For each of these MPI nodes, we have accelerated even more the execution by implementing a GPU version of our application, using CUDA. We can get the best from CUDA only by implementing a clever multi-streaming architecture.
First of all we have created ad hoc memory structures (figure 4.11) At this point, each MPI Node will manage the loading of the poses in the same way as seen in figure 4.11. It is worth noticing that MPI nodes will be responsible for the workload management of each GPU stream. Each node will split the poses he has to process into batches, and then each node sends them to available streams (figure 4.12), the number of poses per batch is defined by a macro, and by default is 256. The goal is to keep streams busy as much time as possible, so it is preferable to tune the application, in such a way to have a lot more batches than streams. So when a stream has finished computation, the node can send another batch to it. Figure 4.13 highlight how GPU threads, blocks, and grids have been organized to deal with our problem. Each blocks i is responsible for the final score of poses i, i*2,... , while each threads t of block i is responsible for retrieving the contribution to the score of atom t, t*2, ..., of the pose that is under process in block i. Since a grid is associated with a stream, and since each stream, a batch of poses is associated,

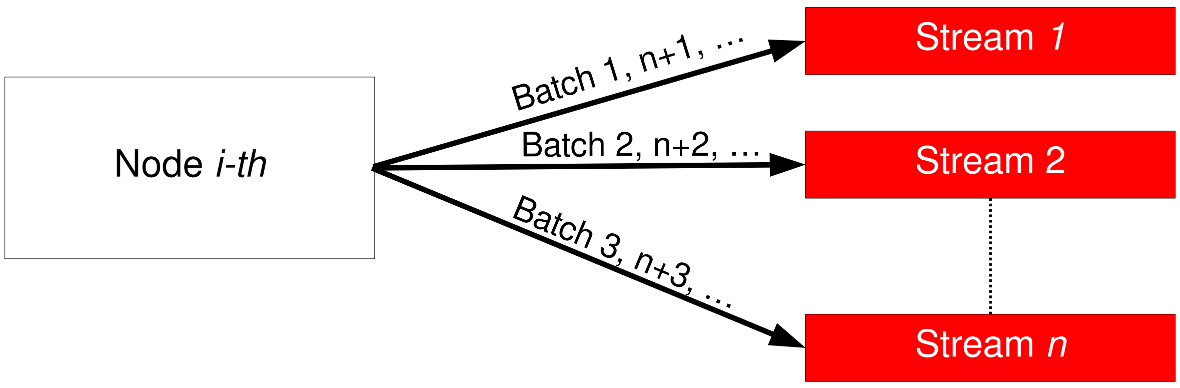


Figure 4.12: MPI CUDA integration architecture.

derived that each block in a grid repetitively processes poses from the streaming batch of poses.

4.2.2. Autodock HB Computation

Pre-computing the HB value using the Autodock algorithm can be done by modifying AutoGrid, and taking only the portion of code responsible for the hydrogen bonds evaluation. Autodock implementation is shipped also with a set of tools, like AutoGrid, from which we extrapolate the HB computation, and a set of scripts that can be used to prepare the input for the Autodock or AutoGrid computation, that can be reused in our case. The pre-computation requires also a pre-parsing stage that assigns to each ligand's atoms its Autodock type.

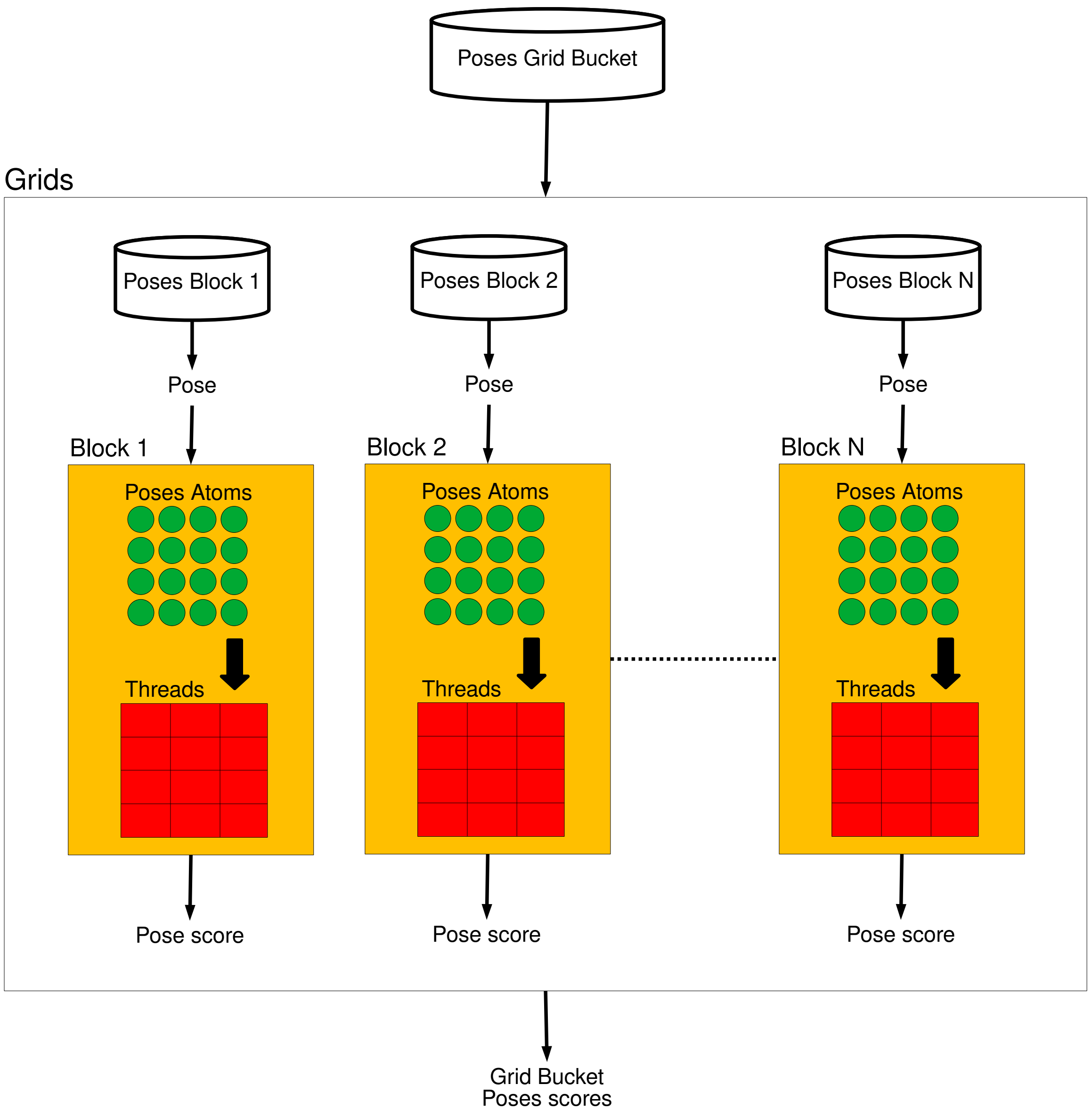


Figure 4.13: GPU threads architecture.

5 | Results Analysis

This chapter describes how we performed the analysis that drove the design of the proposed approach. Moreover, we evaluate the performance when we deploy the filter in different use cases, in particular in Section 5.6. The next sections illustrate the impacts of the approximations used for the Filter development. Approximations are required by the Filter to emulate XSCORE, for being faster, and for increasing the throughput. The structure of this chapter follows the process we have followed when we have started approaching the problem and designing a solution for it. We have started by analyzing how much XSCORE was sensible to precisions scaling, in Section 5.1. At this point, we have checked in Section 5.2 how memoization reduces the accuracy of XSCORE components values. And finally, we have analyzed how a Filter implementation can improve a pipeline, with the implementation integration inside the LiGen pipeline, Section 5.6.

Experimental Set Definition With Test Set, we refer to an experimental dataset of 50 ligand-receptor pairs taken from PDBBind [6].

5.1. Precision Scaling Approximations Analysis

The first approximation we have analyzed was precision scaling. The objective is to test the sensibility of XSCORE to approximations of atom's coordinates and radiuses. Using precision scaling as described in Section 4.1.2 highlights how each atom's coordinates are approximated to their nearest position on the grid. Precision scaling analysis has been conducted mostly on XSCORE factors that have data dependencies on these two parameters (VDW, HM, HP).

5.1.1. Precision Scaling on Coordinates

Coordinates can be quantized by transforming the space from being continuous to being discrete, by snapping an atom to the center of its cube in the grid (see Figure 4.5). In this way, an infinite set of points is reduced to a single point on a grid. In other words, no point of the space can have a distance from another point near to him which is less

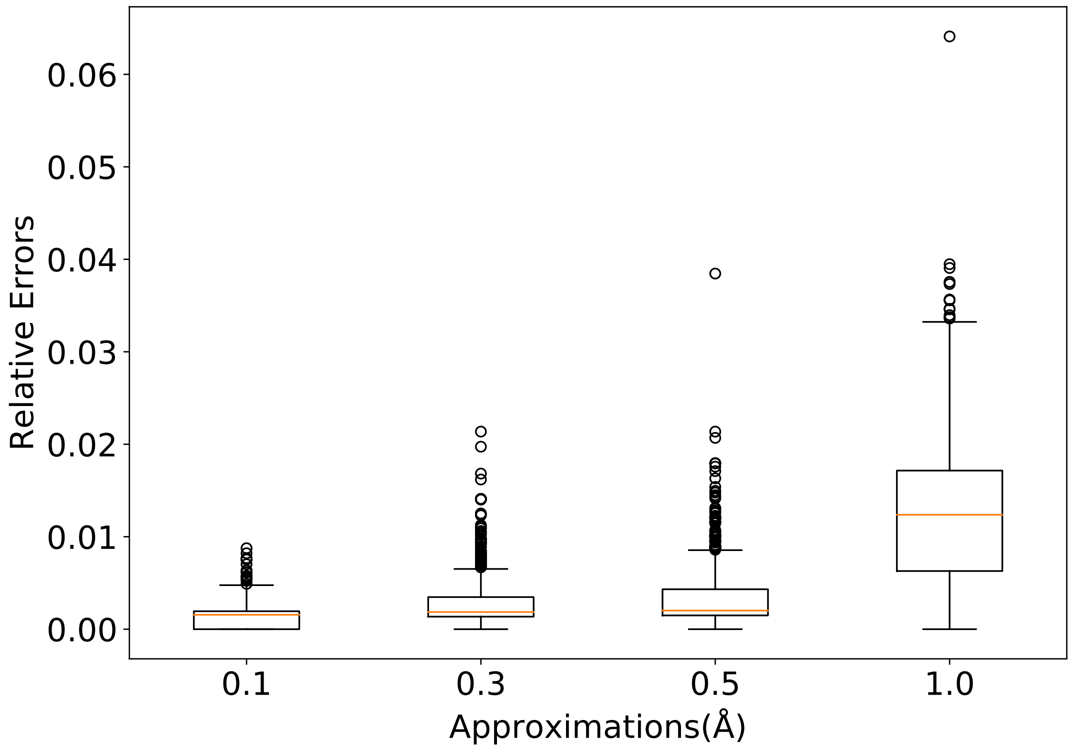


Figure 5.1: Distribution of the XSCORE errors with quantized space. On the y-axis the relative difference between the original values and the quantized ones are reported. While on the x-axis there are different grid resolution.

than the defined spacing along all axes. All atoms' coordinates will be rounded according to their nearest value. Figure 5.1 shows how the error introduced by precision scaling is very restrained: the box plots represent the distribution of the differences between the original XSCORE values, and the XSCORE values obtained when the space is quantized (with a spacing of 0.1, 0.3, 0.5, or 1.0 Å). 0.5 Å is the default spacing value we have used for the thesis since it introduces on average an approximation on the final XSCORE of around 1%.
Higher values have not been considered because the approximation will be too much, and upon pocket analysis, XSCORE is not able to support these rounding. The increase of accuracy when using lower spacing values is not useful if compared with the increase of memory space for memoization (during our analysis the memory requirement is duplicated with triplicate when using a resolution o 0.3 Å, and 10x with a spacing of 0.1 Å) and computational time required for pre-computation (with a spacing of 0.1 Å I was not even able to complete the computation of a single factor). This is because the number of point

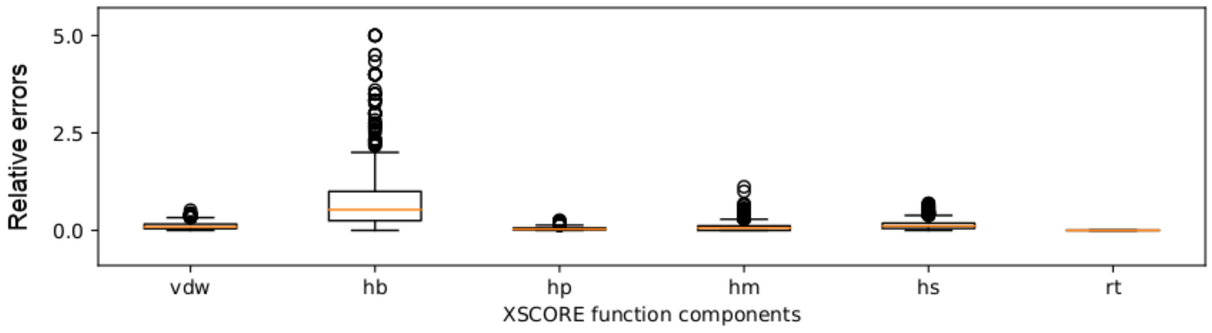


Figure 5.2: Percentage Error on VDW, HB, HP, HM, HS, RT when using precision scaling. The value are capped ot 5%, the outliner above thi threshold are not relevant.

to be analyzed at pre-compute time grows as $\frac{XYZ}{grid{-}spacing^3}$: passing from 0.5 to 0.1 Å grid resolution increase the number of points of 125x. Figure 5.2 reports the relative errors on VDW, HB, HP, HMN, HS, RT (factor of the XSCORE function) when grid's resolution is of 0.5 Å. The figure confirms our suspects that the XSCORE sensibility to precision scaling is very low since on average single component values are impacted for at most of 1%. The only exception comes from the HB factor, which is more sensible to changes in the atoms' coordinates. The content of Figure 5.2 is casted at 5, outliners up to 17 exists but they are not relevant for the average case, so not report.

5.1.2. Precision Scaling on Radius

Precision scaling should be used on radiuses since some XSCORE factors have data dependencies on the atoms' radiuses. Since at the pre-computer time the ligand's atoms are not available, we should discretize the radiuses, to make the input space finite for the pre-computation. Precision scaling on radiuses requires the definition of ranges (a max, min, and avg value for each). By analyzing atoms' definitions used by XSCORE, we can retrieve the maximum and minimum values of radiuses. While from a subset of ligands taken from PDBBind, we have obtained the following radiuses distribution. Figure 5.3 shows the atoms' radiuses distribution inside the test set used for this thesis. Also, atoms with a radius of 0.0 are considered, they are the atom whose atomic type is not supported by the XLOGP [41] list used by XSCORE. Using this figure we got the idea of doing clustering on the most common values of radiuses. Based on the number of ranges used, the loss of accuracy of the results changes. So to decide the number of ranges that best suited our application, we have conducted some tuning of approximation parameters. This is one of the critical parts of approximated computing, that affects a lot the quality and the accuracy of the results. We have analyzed relative errors on VDW, HM, and HP values with different sets of ranges, to understand the consequence of using different sets

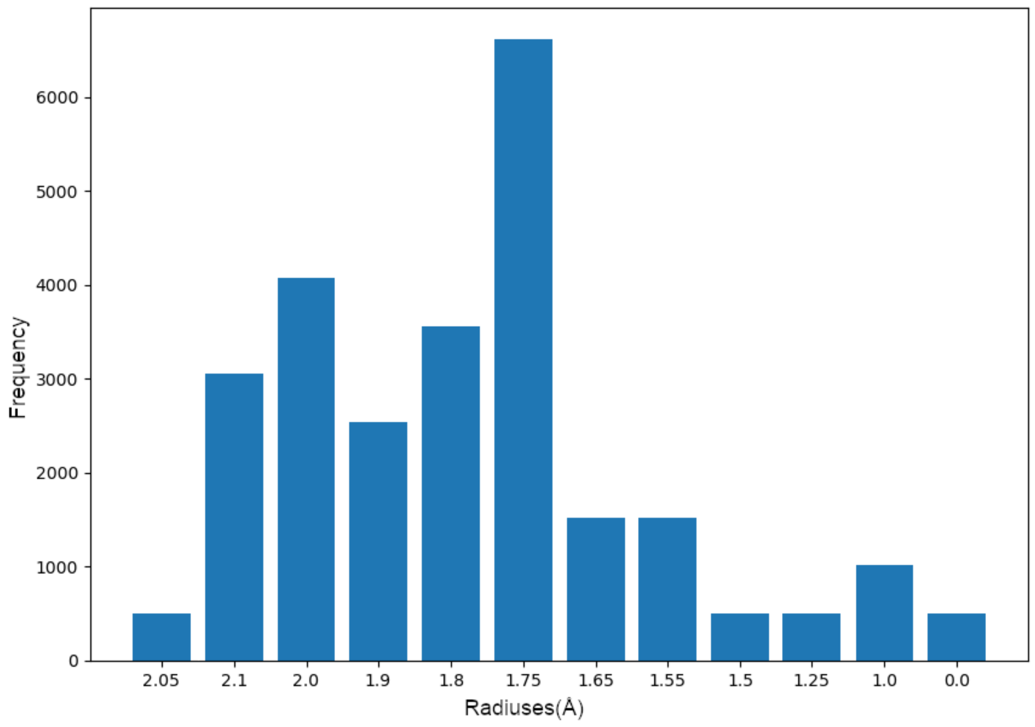


Figure 5.3: Radiuses distribution. On the y-axis the number of occurrences in the PDB-Bind dataset, when varying the atom radius according to the atomic type.

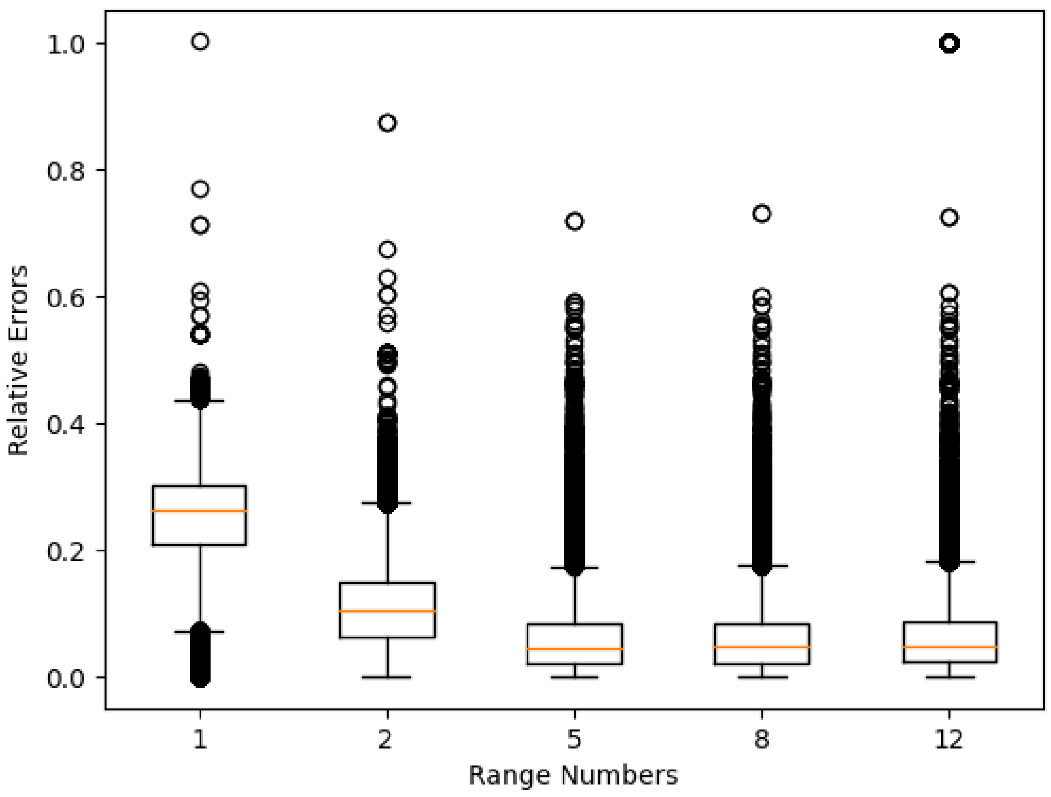


Figure 5.4: VDW box plots of relative errors with sets of ranges with different numbers of clusters. On the x-axis, the number of ranges per set, on the y-axis the relative errors introduces to the VDW values.

of ranges. Figure 5.4 Figure 5.5 Figure 5.6 show that the relative error on VDW, HP and HM values decrease as we increase the number of layers in the set. We have analyzed the loss of accuracy for VDW, HP, and HM factors using sets of ranges of different sizes: 1, 2, 5, 8, and 12 ranges. 12 is the maximum value since from XLOG [41] atoms' type definition there are 12 unique atom radiuses. The analysis in HB has not been done, since its computation is done differently from the radiuses, as explained in Section 4.2.2. The relative errors obtained for each factor on the pair in the test set are reported in the box plots. Is clear from these figures that the decreasing trend reaches a local minimum upon using 5 layers. With 8 and 12 layers, the reduction in the approximation is not worth the increased requirements of memory space and computation time. We have been able to demonstrate that by using a set of 5 ranges, using clusters suggested from Figure 5.3 we are able to obtain a loss of accuracy on the results of around 6% for VDW, 9% HP, and 7%HM.

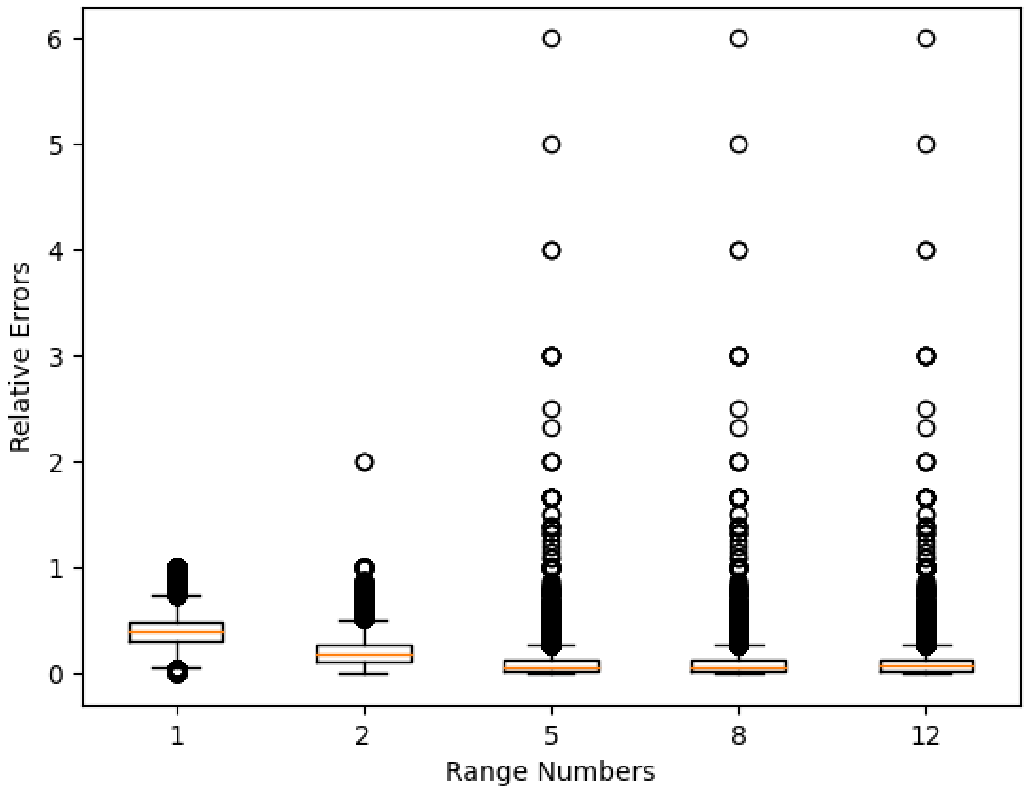


Figure 5.5: VDW box plots of relative errors with sets of ranges with different numbers of clusters. On the x-axis, the number of ranges per set, on the y-axis the relative errors introduces to the HP values.

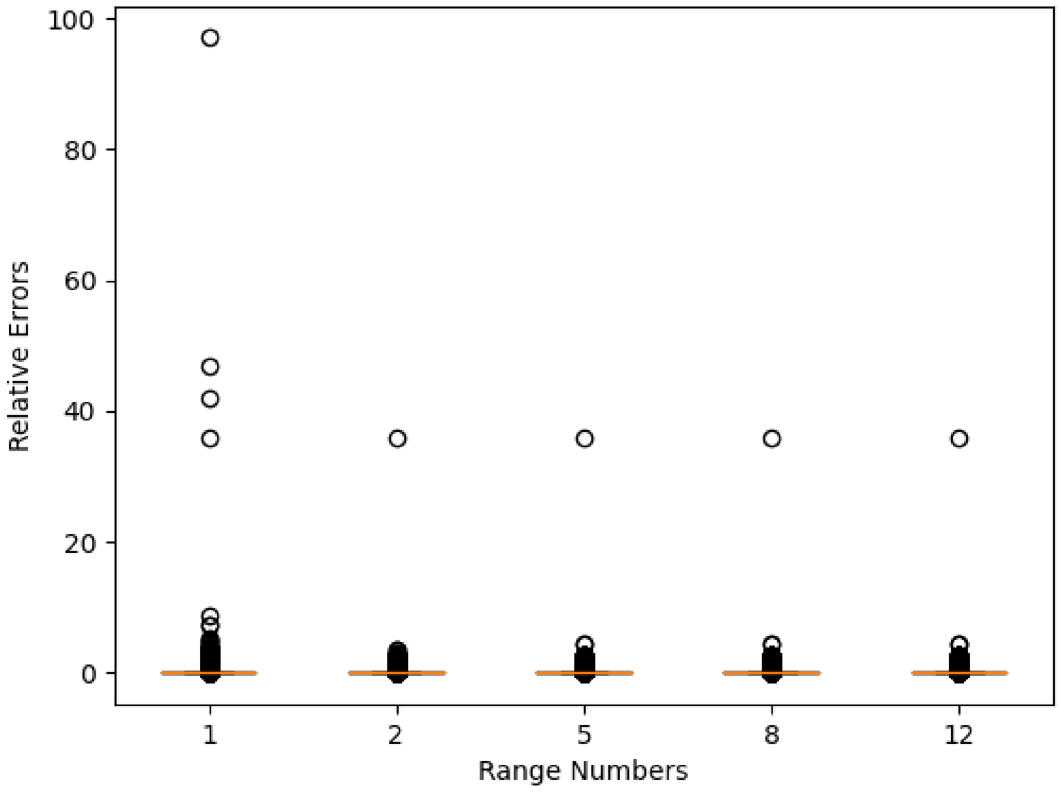


Figure 5.6: VDW box plots of relative errors with sets of ranges with different numbers of clusters. On the x-axis, the number of ranges per set, on the y-axis the relative errors introduces to the HM values.

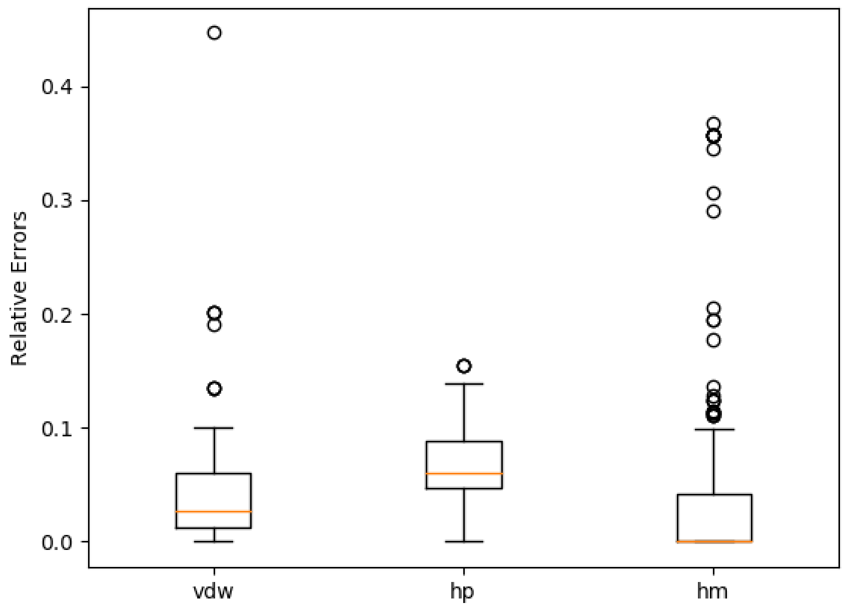


Figure 5.7: Relative errors of VDW, HM, Hp values with precision scaling and memoization. On the y-axis, there is the distribution of the relative errors.

5.2. Memoization and Precision Scaling Approximations Analysis

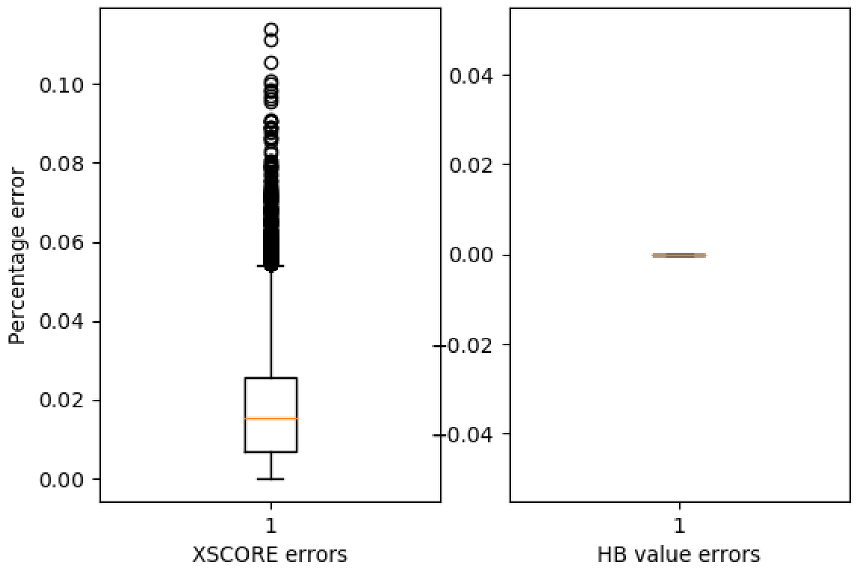
Values errors of Approximated HB wrt to the original. With it ht speed up is of 45x wAfter we have tuned precision scaling, we have evaluated the impact of precisions scaling cooped with memoization. The memoization approximation requires the pre-computation of the XSCORE factors for each possible input. Finally, in this section, we evaluate the final error on XSCORE factors due to precision scaling combined with memoization, when we use the proposed configuration: 0.5 Å grid size and 5 clusters for the layers radiuses. Our goal is to check the impact of precision scaling (grid's spacing of 0.5 Å and 5 radiuses layers). We want to check how memoization impacts those terms which have data dependencies on an atom's radiuses and coordinates. We start with the analysis of VDW, HM, and HP components since they can be pre-computed using an approximated version of the original XSCORE algorithm (HB will be reviewed later on since its approximation is not straightforward Section 4.1.3)

5.2.1. VDW, HM, and HP Memoization and Precision Scaling

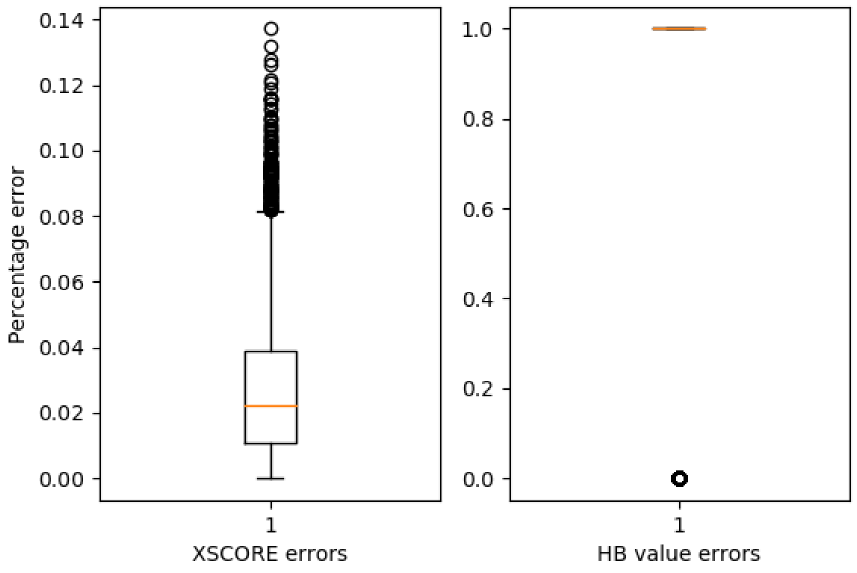
Figure 5.7 shows on the box plot the loss of accuracy on the VDW, HM, and HP values when using precision scaling and memoization together. In particular, the approximation on the VDW values is around 0.3%, on the HP values is around 6%, and on the HM values are around 0%. These are acceptable approximations that can be introduced compared to the gained speed up, from Section 5.5.
The next section shows which of the proposed version in Section 4.1.3 performs better. In fact, in Section 4.1.3 we have introduced the problems encountered in the pre-computation of the HB factor, and different solutions have been proposed.

5.2.2. HB Version Selection and Score Analysis

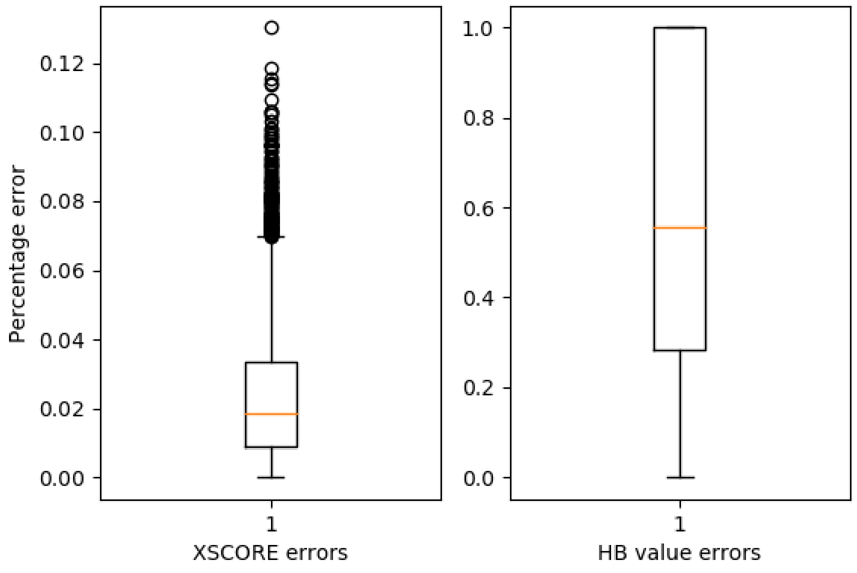
In section 4.1.3 we have reported that the Autodock HB pre-computation is the best-suited approximation for the case of study in this thesis. Taking inspiration from the Auogrid source code shipped with Autodock, we have been able to find out how the pre-computation of the HB term is done by Autodock, and by reusing part of its source code, we were able to integrate it within the demo, reducing the development time, and in this section, we will see also how the accuracy loss has been reduced. Each of the graphs in Figure 5.8 shows two plots: on the left, there is the set of errors on the final score obtained from the Filter on the test set, while on the right there is the set of errors on the HB value obtained with the same Filter version on the same test set. Figure 5.8b since it does not compute the HB values (by default is set to 0), in some cases the error is 100% in others is 0%, because if the real value of HB is 0, then we have done the correct computation casually. From these figures is clear that the version able to reduce the HB values errors is the Full HB version, since its the one that computes the HB factor using the original HB algorithm, without approximation. A trade-off is now analyzed on the approximation: instead of using the Full HB version, we can think about using the Autodock HB, which among the other Filter's versions is the one with the lowest XSCORE and HB errors. In this way, we describe the overall XSCORE error of 0.2%, while the speedup increases from 42x to 47x times. This is why we have chosen to use the Filter with Autodock HB pre-computation as a reference implementation. The Autodock HB speed up is comparable to the one of the No HB Filter version, in this case by using the Autodock HB Filter we reduce the error on the final XSCORE by about 0.5%.
Figure 5.9 shows how a regression can be used to fit the value of HB obtained from the Autodock HB algorithm to HB values computer with XSCORE algorithm.



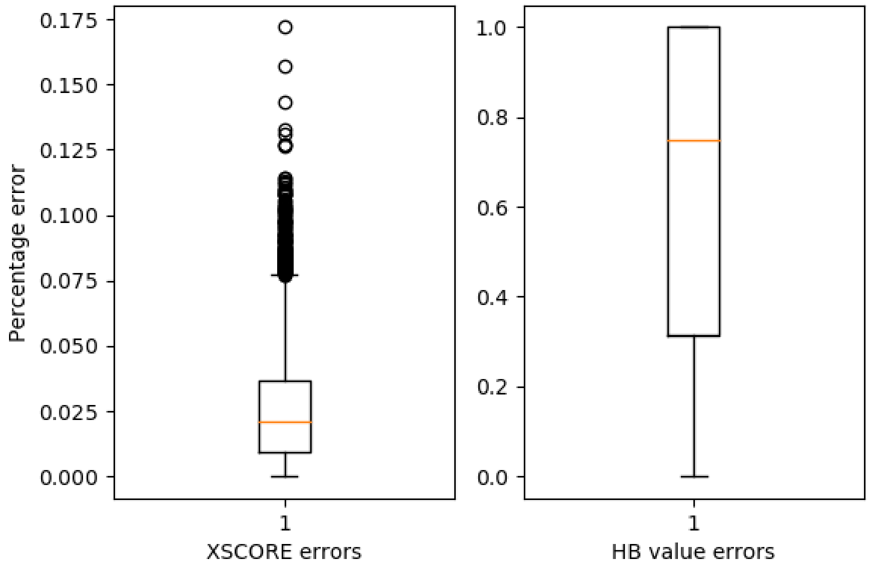
(a) Values errors of Full HB wrt to the original. With it ht speed up is of 43x while the score error is of 1.8%.



(b) Values errors of No HB wrt to the original. With it ht speed up is of 47x while the score error is of 2.8%.



(c) Values errors of Autodock HB wrt to the original. With it ht speed up is of 47x while the score error is of 2.2%.



(d) Values errors of Approximated HB wrt to the original. With it ht speed up is of 45x while the score error is of 2.5%.

Figure 5.8: Values errors with different strategies for approximating the HB component in the Filter block.

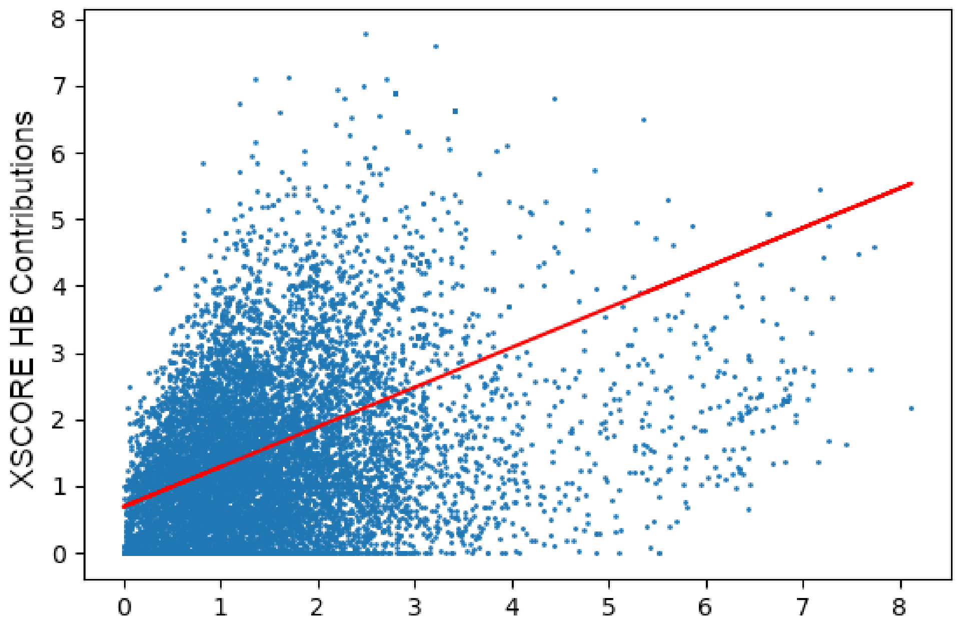


Figure 5.9: Autodock HB Filter version regression wrt to original XSCORE values.

5.3. Simplification Approximation Analysis

Besides precision scaling and memoization we have used other approximations techniques. In particular, the cost of computing a subset of score components, may not be justified by its contribution to the final score value, as shown in Figure 5.10. Therefore, in this section, we evaluate the impact of using those components altogether.
Some parts of the code can be simplified, or are too computationally heavy with respect to their weight on the XSCORE value. In some cases what we introduce as an approximation, has instead no impact on the result accuracy. For example, the apparent charge approximations introduce no error in the XSCORE values.
For example, our analysis has shown how the RT value is the same among all the poses of the same ligand. The deformation effects account for the rotatable bonds present in the ligand, Section 4.1.1. So its impact on the final XSCORE is negligible since it would be the same thing as adding a fixed offset to the XSCORE value for each pose.
A similar analysis to the RT simplification and impact on the score, have been conducted on the HS component. HS computation has been removed as specified in the previous chapter. Figure 5.10 highlights that the loss of accuracy on the final XSCORE value is about 2% on average. While the scoring time is reduced by about 35%.
Regarding apparent charges, we have stated that they can be removed from the computation. For each pose to be scored, XSCORE assigns an apparent charge for each atom, based on its atom type. Since in most of the cases the apparent charge is equal to 0, we

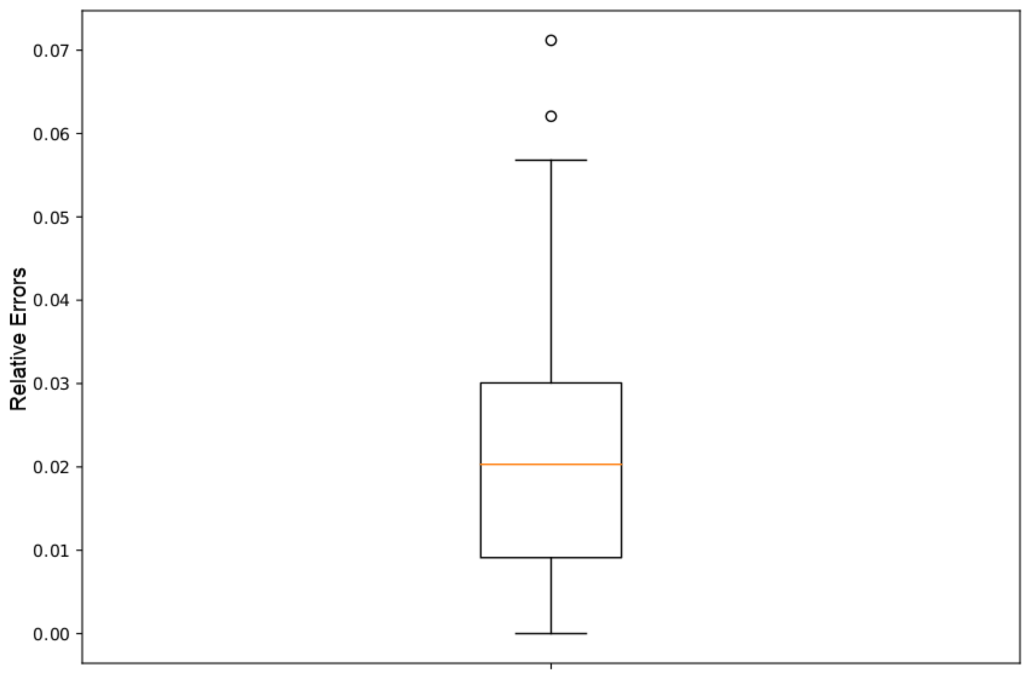


Figure 5.10: XSCORE accuracy loses when removing HSSCORE from the consensus scoring. It is a box plot with the distribution of errors on the final XSCORE values.

have analyzed the impact of removing apparent charges from the computation. The loss of accuracy introduced by removing apparent charges has been computed for each pose of the ligands in the test set. Our hypothesis that removing apparent charges from the computation is a simplification and not an approximation is confirmed by the fact that the average degradation of the results' accuracy is 0%,

5.4. Performance Approximation Results

The previous sections have evaluated the proposed solution as a stand-alone box against a scoring function (XSCORE). In this section, we evaluate the whole approach in two use cases.

5.4.1. RMSDs And Ranking Analysis

The review done in this section has the purpose of showing how we can get in most of the cases the best pose selected by XSCORE and also in the top 10 poses selected by our Filter. And how in general these top 10 approximated poses, have an RMSD that is the same if not better, with respect to the RMSD of the best poses identified by XSCORE. The RMSD is computed in all cases with respect to the crystal pose. Figure 5.11 contains a different graph that compares the original version of XSCORE against the proposed solution with approximations:

- the left plot is how many times in the experimental test set we were able to capture poses with the highest XSCORE returned from the original version, also inside the top 10 poses filtered by our proposed solution;
- the center plot shows the lowest value of RMSD obtained from the 10 best poses selected by our filter against the crystal pose;
- the center plot shows the lowest value of RMSD obtained from the 10 best poses selected by XSCORE against the crystal pose.

The Filter version is using precision scaling with a grid resolution of 5 Å and 5 layers for VDW, HM, and HP pre-computation. While the HB version is pre-computed using the Autodock HB value. In our proposed solution we have used 10 poses for the Filter analysis, since it is much faster with respect to the original XSCORE version, we can then analyze more poses in the same amount of time respect to the original XSCORE version. This analysis is useful if the proposed solution is integrated into the original version as a filter before the scoring function. So only the Filter selected 10 poses to be passed to the original XSCORE version, which now executes on a very small dataset compared with the

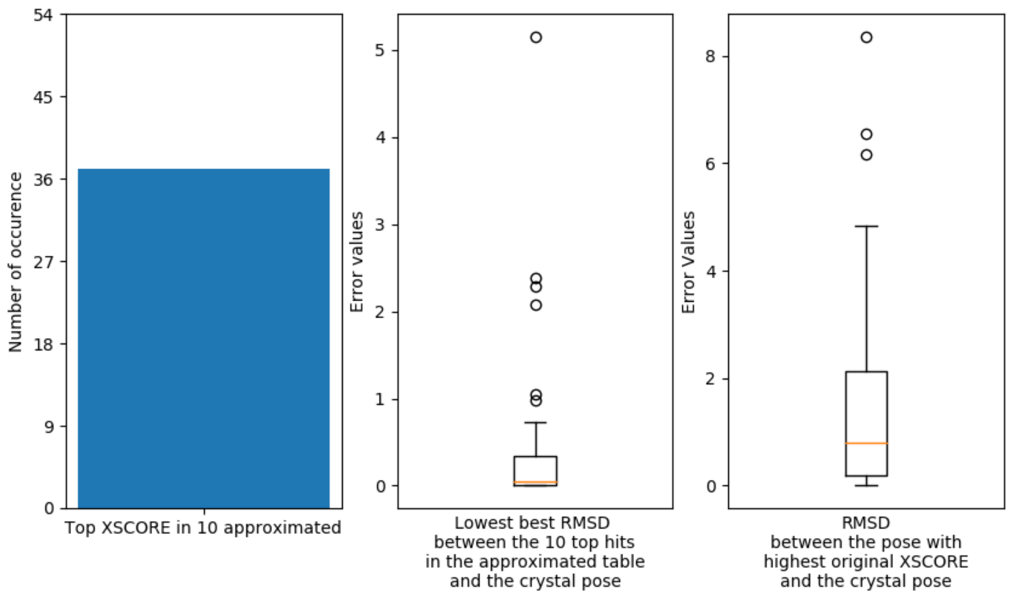


Figure 5.11: RMSD and ranking analysis for the Filter version with Autodock HB pre-computation.

whole set of poses coming as the output of the docking stage. Figure 5.11 indicates how the best RMSD obtained with our Filter is comparable, if not even better, with respect to the RMSD of the best pose identified by XSCORE.

5.4.2. Approximations Impact on Complete Scoring

Is now time to review the impact of all the approximations strategies we have introduces in Section 4.1.2, using different metrics to evaluate the accuracy of the results. To do so we have compared the original XSCORE version, with our proposed version approximated, using a grid's resolution of Å with 5 radiuses layers and hydrogen bond value computed using the Autodock algorithm. The following data have been collected on the experimental dataset defined in Section 5.

Up to this moment, all the analyses have been focused on a single parameter for the approximations evaluations. Is interesting to analyze also the relation there exists among all of the interesting accuracy metrics (RMSD, scoring time, accuracy). For this analysis, we have taken into consideration a single pair of ligand receptors from PDBBind [6], which is 00562_2AY4. For each ligand-receptor pair, PDBBind allows downloading the structure of both receptor and ligand, along with the reference pose of the ligand, which

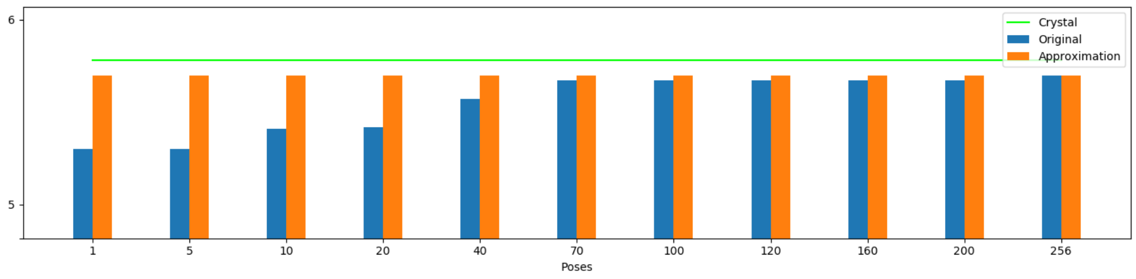


Figure 5.12: Best score comparison when using our approximated pipeline with respect to the original pipeline. on the y-axis the XSCORE values are reported, while on the x-axis we find the number of poses passed as input to the scoring function, for the receptor-ligand pair 00562_2AY4.

is obtained experimentally with crystallography. This pose is called the crystal. The lead compounds we get from the scoring function should have the lower RMSD among all the poses computed from docking programs. Our approximations can increase the number of poses analyzed in the same amount of time. So the possibility of retrieving from the docking program output the pose which has the lower RMSD with respect to the reference crystal is higher. Figure 5.12 on the x-axis has the number of poses passed as input to the scoring function. After the docking program has generated a list of poses in the original pipeline we select only X random poses from this list to be used as input to XSCORE. While in the approximated one we apply our proposed solution as a filtering stage on all the output poses for the original pipeline, and the X first poses from the filtering output of our filter is the input of the original XSCORE's version. The figure clearly shows how also when having a small value for X we can get the best scoring pose, with the approximated version. The approximate version is able to identify almost immediately the best pose because it can scan the whole output poses of the docking stage, while at the same time the original XSCORE version can score only a subset of such output. In Figure 5.12 the green line shows instead the XSCORE value associated with the crystal pose. We are not able to reach this green line with the poses returned by the docking program, due to the limitations of docking algorithms: they cannot usually explore the whole space of poses in order to find the crystal pose. Another graphical representation of the relationships between RMSD, scoring time, and the number of poses is in Figure 5.13. Y-axis on the left refers to the RMSD reported on the bars, while the y-axis on the right refers to the scoring time reported by the two lines. Green bars are the best RMSD between the crystal pose and the poses coming from the output of the docking program, so the RMSD is always the same. Red bars are instead the RMSD between the crystal pose and the best-scored

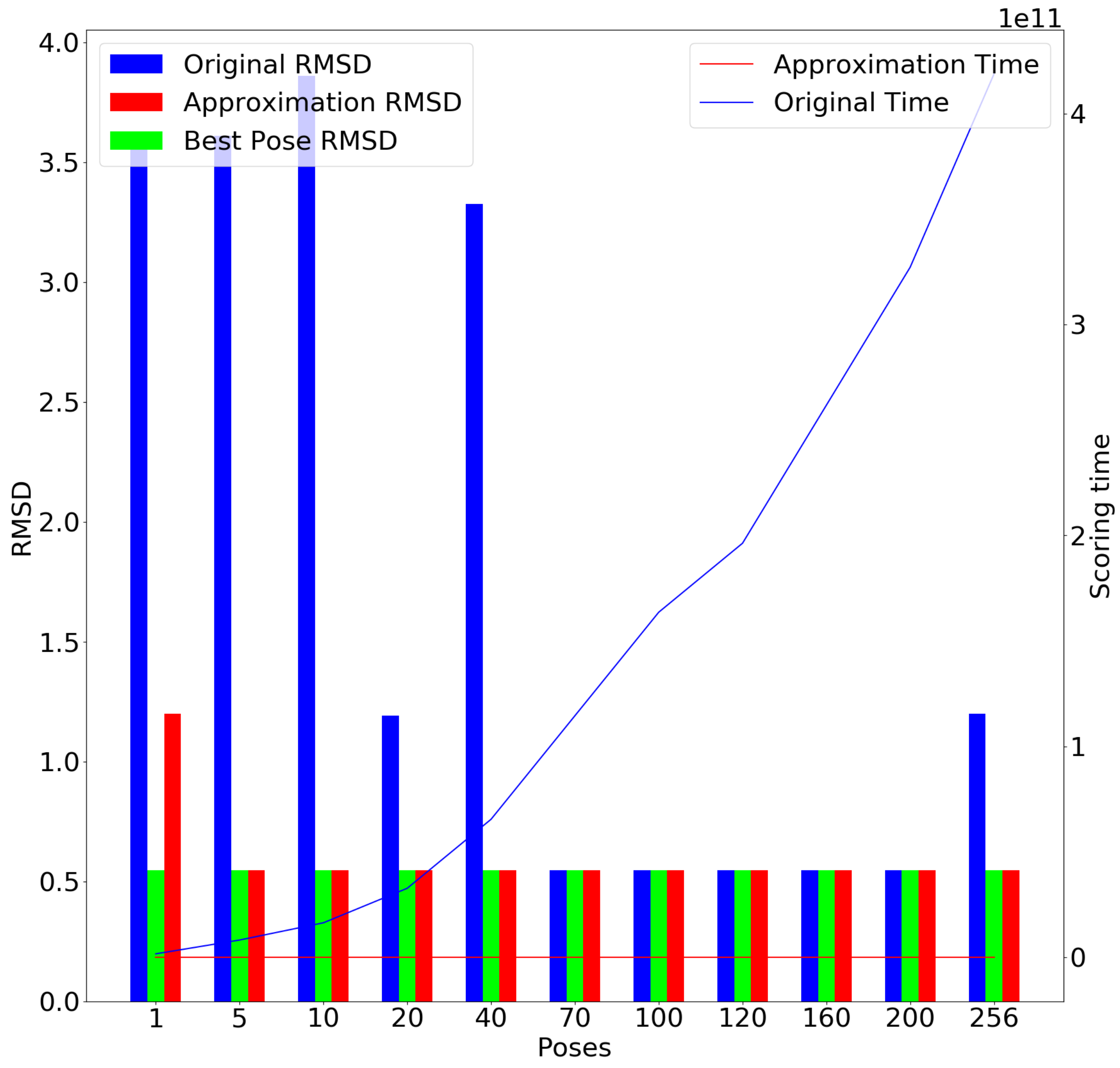


Figure 5.13: Another graphical representation of the relation between RMSD, scoring time, and the number of poses for the receptor-ligand pair 00562_2AY4.

pose, which is the pose that we can get from the original XSCORE version with as input 1, 5, 10, 20, 40, 70, 100, 120, 160, 200, and 256 random poses from the docking program output. The red line is the scoring time of the original XSCORE version. Scoring time of the original version increases as we increase the number of poses passed to XSCORE. Blue bars are the RMSD between the crystal pose and the best-scored pose. In this case, we compare directly the best pose obtained by the filter, without passing from XSCORE, and the Filter is applied on 1, 5, 10, 20, 40, 70, 100, 120, 160, 200, and 256 random poses from the docking program output. The blue line is the scoring time of the Filter, which seems to be a straight line compared to the red line because the Filter's computational complexity increases very slowly with the number of poses if compared to the increase of the original version. Figure 5.13 shows that with the proposed approximated version we are able in a fraction of the time to score more poses with respect to the original XSCORE version. That means that the input space (the docking output) can be explored much faster (as shown by the differences between the red and the blue lines), and at the same time the best poses, with respect to the RMSD value, can be found in a fraction of the original time. In fact, in Figure 5.13 already in the first 5 poses selected by our filter, we are able to identify the poses which have the lower RMSD among all the poses coming from the docking output. With Figure 5.12 Figure 5.13 we have demonstrated how the Filter proposed by this thesis, can select poses with an RMSD and score comparable to the poses selected by XSCORE, but in a fraction of the original time.

5.5. Speedup: CUDA and MPI vs CPU

Part of the thesis development has also been focused on a Filter version for HPC, implementing all the approximations strategies we have seen in Section 4.1.2. The previous chapter has better detailed 4.2.1 the CUDA and MPI version of the Filter. Since we are analyzing the impact of approximations, we want also to include in this analysis the importance of using MPI and CUDA had for the Filter implementation, when compared with the original XSCORE version. Figure 5.14 reports on the y-axis the occurrences of speedups on the scoring time, between the HPC Filter version against the single CPU XSCORE version. The cited figure highlights the advantages of using the HPC version of the Filter, which has on average a speedup on the computation time of around 26x, which is also confirmed by the speedup distribution in the same figure. The values distribution we can see in Figure 5.14 confirms also as the variance is acceptable: most of the speed-up occupancies are between 10x and 30x, which is a very great improvement for the throughput of a virtual screening application.

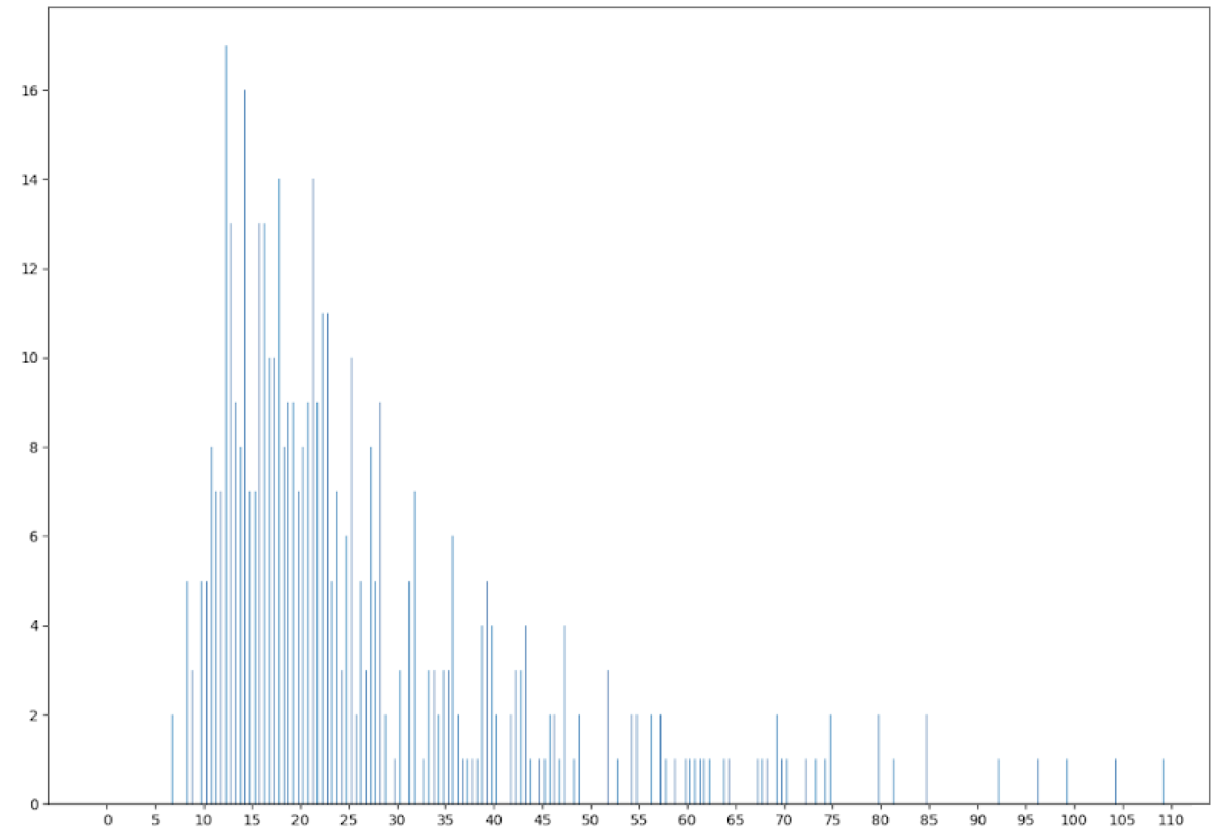


Figure 5.14: Speedup of using the GPU version of the Filter with respect to the original only CPU XSCORE version.

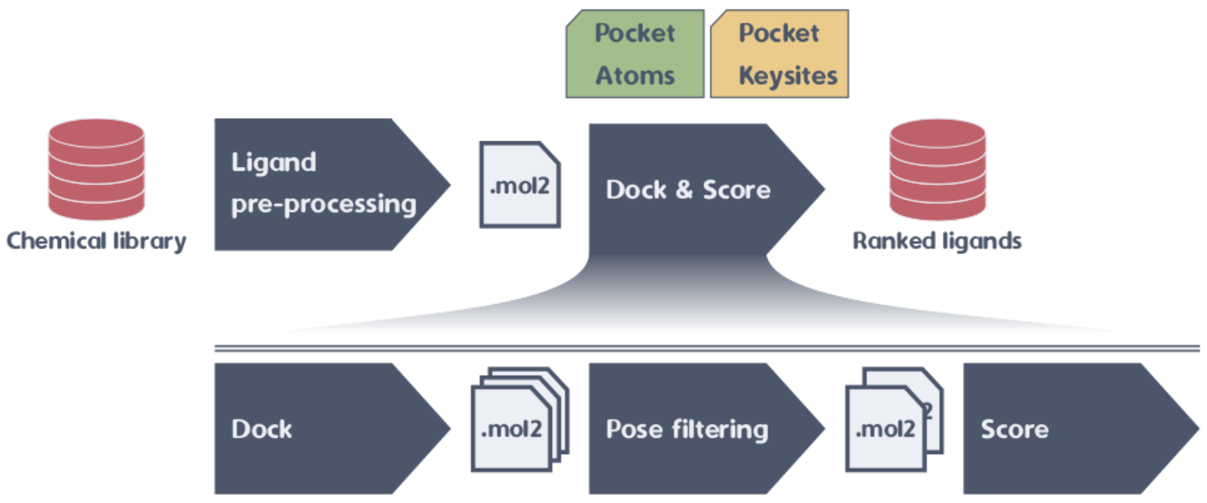


Figure 5.15: Ligen Pipeline, taken from the LiGen doc.

5.6. LiGen Pipeline Integration Results

Figure 5.12 Figure 5.13 demonstrate how the Filter performs when compared against a scoring function. We have already stated how the Filter can be used in different ways: on one hand, it can be used to evaluate the same number of poses with respect to the original version, but in a fraction of time, or on the other hand it can be executed in the same amount of time on more poses, increasing the throughput of the docking program. In conclusion, the figures confirm the Filter capacity of identifying the poses with both a lower RMSD and a higher score, in a smaller amount of time, when compared with the original XSCORE version: we have been able to confirm that the throughput of a virtual screening pipeline for drug discovery can be increased using approximated computing techniques.
Using the same Filter is now time to review its performance when it is inserted inside a real-world virtual screening pipeline. In this section in fact we analyze the results obtained when inserting the Filter inside the LiGen pipeline.

5.6.1. LiGen Pipeline

The driven concept of Ligen's architecture is the concept of OPA or *one-purpose application*. The workflow is shown in Figure 5.15: LiGen is composed of small applications with a specific scope and meaning, which are chained altogether. This is a very good feature in our case: in fact, we can more easily put our Filter inside the workflow of LiGen. I've modified the LiGen pipeline, Figure 5.15, which as in Figure 4.9 has already a Pose Filtering stage, which uses some quick heuristic. In this case, I've substituted it with the proposed Filter of this thesis, and I've added the pre-computation stage to the pipeline, which has to be executed once for each receptor under analysis.

5.6.2. Pose Filtering and Pre-computation

The Filter I've integrated inside the pipeline uses a grid's resolution of 0.5 Å and a number of 5 layers (based on radiuses), the HB value is computed using the Autodock algorithm. The implementation of the Filter is done using CUDA and MPI as stack technologies. For the data analysis, I've taken 10 randomly receptor-ligand pairs from the experimental dataset (Section 5). From the previous results, we expect to decrease the execution time of the pipeline, thus increasing the throughput, while the results' accuracy loss should be a low value.
Referring to Figure 5.15 we do make the following statement on how our approximation works:

- *Only Scoring* considers the Ligen version in which the Filter is ignored by the workflow, and so all the output poses of LiGen are processed by XSCORE;
- *Filtering+Scoring* considers the LiGen's pipeline version with the Filter, which selects only the best 15 poses to be scored by the scoring function;
- With RMSD we intend the RMSD of the best scored pose.

The collected data are reported in Table 5.1 The table highlight how the insertion of the Filter into the workflow makes the computation faster, while the best RMSD remains the same. This means that our hypothesis is confirmed, and the LiGen pipeline can increase its throughput by using the proposed Filter of this thesis, instead of the quick heuristics already used in the original Pose Filtering Stage.

Protein Name	**Only Scoring**		**Filtering+Scoring**		**Speedup**
	Time (ms)	Best RMSD (Å)	Time (ms)	Best RMSD (Å)	
1a0q	3076	3.04	240	3.04	12.8
1a0t	3588	5.6	312	5.6	11.5
1a1b	5322	10.6	401	10.6	13.2
1a3e	13065	7.27	1315	7.27	9.93
1a4h	5926	1.3	598	1.08	9.9
1a5g	7025	2.99	690	2.99	10.4
1a5h	4726	2.6	399	2.6	11.8
1a07	4748	5.7	381	5.7	12.4
1a7x	10832	2.85	2124	2.85	5.09
1a08	5925	10.7	438	10.7	13.5

Table 5.1: This tables illustrate the obtained results with the proposed solution applied to the LiGen pipeline. It compares the RMSD and time obtained with the original pipeline version, and with the pipeline modified with the proposed solution.

6 | Conclusion

The drug discovery process is a long and costly process [44]. Computer-generated models, to dock and score molecules, have been applied to the drug discovery process, leading to the implementation of a virtual screening pipeline. At this point, the analysis of compounds is done in virtual environments by supercomputers, not only in vitro by researchers. Thus high efficiency and small time-to-solutions are key requirements for the virtual screening pipeline. The solution I've proposed is throughput-oriented. To increase the throughput I've decided to analyze the impact of approximated computing on a virtual screening pipeline. This thesis has dissected the problem of increasing the throughput of a virtual screening pipeline in detail, with the objectives of proposing, implementing, and analyzing a solution. Starting from the problem analysis, I've analyzed how a virtual screening pipeline is used in the drug discovery process. The stage on which the thesis has focused most of its attention is the scoring stage. In Chapter 4 the scoring stage has been disassembled, with a particular interest in XSCORE. Based on this knowledge, this document explains a way in which precision scaling and memoization can be used to approximate the scoring stage. Using these approximation strategies the thesis was able to anticipate most of the computation at pre-compute time. The docked poses coming from the docking stage have to be scored, and a first evaluation is done thanks to the pre-computed data, to avoid the burden of doing all the computation during a run of a virtual screening pipeline. Wrapping up the results we have gathered and reported into Chapter 5, the proposed solution has proven to be effective in increasing the throughput of a virtual screening pipeline, with a negligible loss in accuracy.
The proposed solution obtained good results, both in the analysis of precision scaling and memoization impact on each component of the scoring function and on the final value of the score. With these results, the thesis has continued its analysis to check the solution's impact on a real case scenario: LiGen's pipeline. In the end with LiGen, we were able to obtain an average speedup of 10x Section 5.6, while the error is almost 0%.

Future Developments The proposed solution can further analyze the impact of other approximations techniques, for example, this document has not considered the impact of

a neural network accelerator, which can be useful for the factors' coefficients of a scoring function. Other work of optimization could be done on the final implemented version of the Filter. The implemented code can be further optimized, and the pre-computation stage can be better organized. Especially the AutoGrid computation can be rewritten and inserted directly into the whole pre-computation natively, without using the original Autodock source code. The implementation used in this thesis used OpenMPI and CUDA. Other technologies can be used to extend the support on other architectures and hardware (for example SYCL). On the actual implementation, a deeper analysis of the usage of GPU's streams and GPU's memory management should be done, which can lead to great improvement in the application throughput.
The work illustrated in this thesis can be further expanded. Its validity should be tested on another pipeline, so also other scoring functions should be taken into consideration.

6.1. Acknowledgements

For the work done in this thesis, I wanted to thank the DEIB department at Politecnico Of Milan, for all the support I've received, especially the willingness of professor Gianluca Palermo, Davide Gadioli, Alfonso Gauteri, and Emanuale Vitali during the development of the thesis.

Bibliography

[1] Núria López, Luigi Del Debbio, Marc Baaden, Matej Praprotnik, Laura Grigori, Catarina Simões, Serge Bogaerts, Florian Berberich, Thomas Lippert, Janne Ignatius, Philippe Lavocat, Oriol Pineda, Maria Grazia Giuffreda, Sergi Girona, Dieter Kranzlmüller, Michael M. Resch, Gabriella Scipione, and Thomas Schulthess. Lessons learned from urgent computing in europe: Tackling the COVID-19 pandemic. *Proceedings of the National Academy of Sciences*, 118(46), November 2021. doi: 10.1073/pnas.2024891118. URL `https://doi.org/10.1073/pnas.2024891118`.

[2] Exscalate4cov project. `https://www.exscalate4cov.eu/index.html`.

[3] A. Acharya, R. Agarwal, M. B. Baker, J. Baudry, D. Bhowmik, S. Boehm, K. G. Byler, S. Y. Chen, L. Coates, C. J. Cooper, O. Demerdash, I. Daidone, J. D. Eblen, S. Ellingson, S. Forli, J. Glaser, J. C. Gumbart, J. Gunnels, O. Hernandez, S. Irle, D. W. Kneller, A. Kovalevsky, J. Larkin, T. J. Lawrence, S. LeGrand, S.-H. Liu, J.C. Mitchell, G. Park, J.M. Parks, A. Pavlova, L. Petridis, D. Poole, L. Pouchard, A. Ramanathan, D. M. Rogers, D. Santos-Martins, A. Scheinberg, A. Sedova, Y. Shen, J. C. Smith, M. D. Smith, C. Soto, A. Tsaris, M. Thavappiragasam, A. F. Tillack, J. V. Vermaas, V. Q. Vuong, J. Yin, S. Yoo, M. Zahran, and L. Zanetti-Polzi. Supercomputer-based ensemble docking drug discovery pipeline with application to covid-19. *Journal of Chemical Information and Modeling*, 60 (12):5832–5852, December 2020. doi: 10.1021/acs.jcim.0c01010. URL `https://doi.org/10.1021/acs.jcim.0c01010`.

[4] Zhihai Liu, Yan Li, Li Han, Jie Li, Jie Liu, Zhixiong Zhao, Wei Nie, Yuchen Liu, and Renxiao Wang. PDB-wide collection of binding data: current status of the PDBbind database. *Bioinformatics*, 31(3):405–412, October 2014. doi: 10.1093/bioinformatics/btu626. URL `https://doi.org/10.1093/bioinformatics/btu626`.

[5] Hrishav Bakul Barua and Kartick Chandra Mondal. Approximate computing: A survey of recent trends—bringing greenness to computing and communication. *Journal of The Institution of Engineers (India): Series B*, 100(6):619–626, June 2019. doi: 10.1007/s40031-019-00418-8. URL `https://doi.org/10.1007/s40031-019-00418-8`.

[6] Pdbbind. http://www.pdbbind.org.cn/.

[7] Pymol. https://pymol.org/2/.

[8] JP Hughes, S Rees, SB Kalindjian, and KL Philpott. Principles of early drug discovery. *British Journal of Pharmacology*, 162(6):1239–1249, February 2011. doi: 10.1111/j.1476-5381.2010.01127.x. URL https://doi.org/10.1111/j.1476-5381.2010.01127.x.

[9] Veronica Salmaso and Stefano Moro. Bridging molecular docking to molecular dynamics in exploring ligand-protein recognition process: An overview. *Frontiers in Pharmacology*, 9, August 2018. doi: 10.3389/fphar.2018.00923. URL https://doi.org/10.3389/fphar.2018.00923.

[10] Jin Li, Ailing Fu, and Le Zhang. An overview of scoring functions used for protein–ligand interactions in molecular docking. *Interdisciplinary Sciences: Computational Life Sciences*, 11(2):320–328, March 2019. doi: 10.1007/s12539-019-00327-w. URL https://doi.org/10.1007/s12539-019-00327-w.

[11] Baohua Zhang, Hui Li, Kunqian Yu, and Zhong Jin. Molecular docking-based computational platform for high-throughput virtual screening. *CCF Transactions on High Performance Computing*, January 2022. doi: 10.1007/s42514-021-00086-5. URL https://doi.org/10.1007/s42514-021-00086-5.

[12] Top 500 supercomputer. https://www.top500.org/lists/top500/.

[13] Mark von Itzstein, Wen-Yang Wu, Gaik B. Kok, Michael S. Pegg, Jeffrey C. Dyason, Betty Jin, Tho Van Phan, Mark L. Smythe, Hume F. White, Stuart W. Oliver, Peter M. Colman, Joseph N. Varghese, D. Michael Ryan, Jacqueline M. Woods, Richard C. Bethell, Vanessa J. Hotham, Janet M. Cameron, and Charles R. Penn. Rational design of potent sialidase-based inhibitors of influenza virus replication. *Nature*, 363(6428):418–423, June 1993. doi: 10.1038/363418a0. URL https://doi.org/10.1038/363418a0.

[14] Davide Gadioli, Gianluca Palermo, Stefano Cherubin, Emanuele Vitali, Giovanni Agosta, Candida Manelfi, Andrea R. Beccari, Carlo Cavazzoni, Nico Sanna, and Cristina Silvano. Tunable approximations to control time-to-solution in an HPC molecular docking mini-app. *The Journal of Supercomputing*, 77(1):841–869, April 2020. doi: 10.1007/s11227-020-03295-x. URL https://doi.org/10.1007/s11227-020-03295-x.

[15] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Don-

garra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104. Springer Berlin Heidelberg, 2004. doi: 10.1007/978-3-540-30218-6_19. URL `https://doi.org/10.1007/978-3-540-30218-6_19`.

[16] Cuda references. `https://developer.nvidia.com/cuda-zone`.

[17] Approximated computing review. `https://co-design.t.u-tokyo.ac.jp/en/research/approximate-computing/`.

[18] Nataraj S. Pagadala, Khajamohiddin Syed, and Jack Tuszynski. Software for molecular docking: a review. *Biophysical Reviews*, 9(2):91–102, January 2017. doi: 10.1007/s12551-016-0247-1. URL `https://doi.org/10.1007/s12551-016-0247-1`.

[19] Israel T. Desta, Kathryn A. Porter, Bing Xia, Dima Kozakov, and Sandor Vajda. Performance and its limits in rigid body protein-protein docking. *Structure*, 28(9): 1071–1081.e3, September 2020. doi: 10.1016/j.str.2020.06.006. URL `https://doi.org/10.1016/j.str.2020.06.006`.

[20] José Ignacio Garzon, José Ramón Lopéz-Blanco, Carles Pons, Julio Kovacs, Ruben Abagyan, Juan Fernandez-Recio, and Pablo Chacon. FRODOCK: a new approach for fast rotational protein–protein docking. *Bioinformatics*, 25(19):2544–2551, July 2009. doi: 10.1093/bioinformatics/btp447. URL `https://doi.org/10.1093/bioinformatics/btp447`.

[21] B. G. Pierce, K. Wiehe, H. Hwang, B.-H. Kim, T. Vreven, and Z. Weng. ZDOCK server: interactive docking prediction of protein-protein complexes and symmetric multimers. *Bioinformatics*, 30(12):1771–1773, February 2014. doi: 10.1093/bioinformatics/btu097. URL `https://doi.org/10.1093/bioinformatics/btu097`.

[22] Andrea R. Beccari, Carlo Cavazzoni, Claudia Beato, and Gabriele Costantino. LiGen: A high performance workflow for chemistry driven de novo design. *Journal of Chemical Information and Modeling*, 53(6):1518–1527, May 2013. doi: 10.1021/ci400078g. URL `https://doi.org/10.1021/ci400078g`.

[23] Todd J.A. Ewing, Shingo Makino, A. Geoffrey Skillman, and Irwin D. Kuntz. *Journal of Computer-Aided Molecular Design*, 15(5):411–428, 2001. doi: 10.1023/a:1011115820450. URL `https://doi.org/10.1023/a:1011115820450`.

[24] Garrett M. Morris, David S. Goodsell, Robert S. Halliday, Ruth Huey, William E.

Hart, Richard K. Belew, and Arthur J. Olson. Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry*, 19(14):1639–1662, November 1998. doi: 10.1002/(sici)1096-987x(19981115)19:14<1639::aid-jcc10>3.0.co;2-b. URL `https://doi.org/10.1002/(sici)1096-987x(19981115)19:14<1639::aid-jcc10>3.0.co;2-b`.

[25] Sheng-You Huang, Sam Z. Grinter, and Xiaoqin Zou. Scoring functions and their evaluation methods for protein–ligand docking: recent advances and future directions. *Physical Chemistry Chemical Physics*, 12(40):12899, 2010. doi: 10.1039/c0cp00151a. URL `https://doi.org/10.1039/c0cp00151a`.

[26] Jie Liu and Renxiao Wang. Classification of current scoring functions. *Journal of Chemical Information and Modeling*, 55(3):475–482, February 2015. doi: 10.1021/ci500731a. URL `https://doi.org/10.1021/ci500731a`.

[27] Patrick Pfeffer and Holger Gohlke. DrugScoreRNAKnowledge-based scoring function to predict RNA-ligand interactions. *Journal of Chemical Information and Modeling*, 47(5):1868–1876, August 2007. doi: 10.1021/ci700134p. URL `https://doi.org/10.1021/ci700134p`.

[28] Matthew D. Eldridge, Christopher W. Murray, Timothy R. Auton, Gaia V. Paolini, and Roger P. Mee. *Journal of Computer-Aided Molecular Design*, 11(5):425–445, 1997. doi: 10.1023/a:1007996124545. URL `https://doi.org/10.1023/a:1007996124545`.

[29] Renxiao Wang, Liang Liu, Luhua Lai, and Youqi Tang. SCORE: A new empirical method for estimating the binding affinity of a protein-ligand complex. *Journal of Molecular Modeling*, 4(12):379–394, December 1998. doi: 10.1007/s008940050096. URL `https://doi.org/10.1007/s008940050096`.

[30] Renxiao Wang, Luhua Lai, and Shaomeng Wang. *Journal of Computer-Aided Molecular Design*, 16(1):11–26, 2002. doi: 10.1023/a:1016357811882. URL `https://doi.org/10.1023/a:1016357811882`.

[31] Paul S. Charifson, Joseph J. Corkery, Mark A. Murcko, and W. Patrick Walters. Consensus scoring: a method for obtaining improved hit rates from docking databases of three-dimensional structures into proteins. *Journal of Medicinal Chemistry*, 42(25):5100–5109, November 1999. doi: 10.1021/jm990352k. URL `https://doi.org/10.1021/jm990352k`.

[32] Ruth Huey, David Goodsell, Garrett Morris, and Arthur Olson. Grid-based hydrogen bond potentials with improved directionality. *Letters in Drug Design & Discovery*,

1(2):178–183, April 2004. doi: 10.2174/1570180043485581. URL `https://doi.org/10.2174/1570180043485581`.

[33] Kaushik Roy and Anand Raghunathan. Approximate computing: An energy-efficient computing technique for error resilient applications. In *2015 IEEE Computer Society Annual Symposium on VLSI*. IEEE, July 2015. doi: 10.1109/isvlsi.2015.130. URL `https://doi.org/10.1109/isvlsi.2015.130`.

[34] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4):1–33, May 2016. doi: 10.1145/2893356. URL `https://doi.org/10.1145/2893356`.

[35] Davide Gadioli, Emanuele Vitali, Gianluca Palermo, and Cristina Silvano. margot: A dynamic autotuning framework for self-aware approximate computing. *IEEE Transactions on Computers*, 68(5):713–728, 2019. doi: 10.1109/TC.2018.2883597.

[36] Arnab Raha and Vijay Raghunathan. q LUT. *ACM Transactions on Embedded Computing Systems*, 16(5s):1–23, October 2017. doi: 10.1145/3126531. URL `https://doi.org/10.1145/3126531`.

[37] Autodock: Grid maps. `http://www.csb.yale.edu/userguides/datamanip/autodock/html/Using_AutoDock_305.9.html`.

[38] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI message passing interface standard. In *Programming Environments for Massively Parallel Distributed Systems*, pages 213–218. Birkhäuser Basel, 1994. doi: 10.1007/978-3-0348-8534-8_21. URL `https://doi.org/10.1007/978-3-0348-8534-8_21`.

[39] Openmpi references. `https://www.open-mpi.org/`.

[40] Mpich. `https://www.mpich.org/`.

[41] Renxiao Wang, Ying Gao, and Luhua Lai. Xlogp. *Perspectives in Drug Discovery and Design*, 19(1):47–66, 2000. doi: 10.1023/a:1008763405023. URL `https://doi.org/10.1023/a:1008763405023`.

[42] Hans-Joachim Bohm. The development of a simple empirical scoring function to estimate the binding constant for a protein-ligand complex of known three-dimensional structure. *Journal of Computer-Aided Molecular Design*, 8(3):243–256, June 1994. doi: 10.1007/bf00126743. URL `https://doi.org/10.1007/bf00126743`.

[43] Daniel F. Veber, Stephen R. Johnson, Hung-Yuan Cheng, Brian R. Smith, Keith W. Ward, and Kenneth D. Kopple. Molecular properties that influence the oral bioavail-

ability of drug candidates. *Journal of Medicinal Chemistry*, 45(12):2615–2623, May 2002. doi: 10.1021/jm020017n. URL `https://doi.org/10.1021/jm020017n`.

[44] Steve Morgan, Paul Grootendorst, Joel Lexchin, Colleen Cunningham, and Devon Greyson. The cost of drug development: A systematic review. *Health Policy*, 100(1): 4–17, April 2011. doi: 10.1016/j.healthpol.2010.12.002. URL `https://doi.org/10.1016/j.healthpol.2010.12.002`.