Scalable HPC & AI Infrastructure for COVID-19 Therapeutics

Hyungro Lee
Rutgers University
Piscataway, New Jersey, USA
hyungro.lee@rutgers.edu

Andre Merzky
Rutgers University
Piscataway, New Jersey, USA
andre@merzky.net

Li Tan
Brookhaven National Laboratory
Upton, New York, USA
ltan@bnl.gov

Mikhail Titov
Rutgers University
Piscataway, New Jersey, USA
mikhail.titov@rutgers.edu

Matteo Turilli
Rutgers University
Piscataway, New Jersey, USA
matteo.turilli@rutgers.edu

Dario Alfe
University College London
London, United Kingdom
d.alfe@ucl.ac.uk

Agastya Bhati
University College London
London, United Kingdom
agastya.bhati.14@ucl.ac.uk

Alex Brace
Argonne National Laboratory
Lemont, Illinois, USA
abrace@anl.gov

Austin Clyde
University of Chicago
Chicago, Illinois, USA
aclyde@uchicago.edu

Peter Coveney
University College London
London, United Kingdom
p.v.coveney@ucl.ac.uk

Heng Ma
Argonne National Laboratory
Lemont, Illinois, USA
heng.ma@anl.gov

Arvind Ramanathan
Argonne National Laboratory
Lemont, Illinois, USA
ramanathana@anl.gov

Rick Stevens
University of Chicago
Chicago, Illinois, USA
stevens@anl.gov

Anda Trifan
University of Illinois at
Urbana-Champaign
Champaign, Illinois, USA
atrifan2@illinois.edu

Hubertus Van Dam
Brookhaven National Laboratory
Upton, New York, USA
hvandam@bnl.gov

Shunzhou Wan
University College London
London, United Kingdom
shunzhou.wan@ucl.ac.uk

Sean Wilkinson
Oak Ridge National Laboratory
Knoxville, Tennessee, USA
wilkinsonsr@ornl.gov

Shantenu Jha
Brookhaven National Laboratory
Upton, New York, USA
Rutgers University
Piscataway, New Jersey, USA
shantenu@bnl.gov

Abstract

COVID-19 has claimed more than 2.7×10^6 lives and resulted in over 124×10^6 infections. There is an urgent need to identify drugs that can inhibit SARS-CoV-2. We discuss innovations in computational infrastructure and methods that are accelerating and advancing drug design. Specifically, we describe several methods that integrate artificial intelligence and simulation-based approaches, and the design of computational infrastructure to support these methods at scale. We discuss their implementation, characterize their performance, and highlight science advances that these capabilities have enabled.

PASC '21, July 5–9, 2021, Geneva, Switzerland

ACM ISBN 978-1-4503-8563-3/21/07…$15.00
https://doi.org/10.1145/3468267.3470573

CCS Concepts

• **Computing methodologies** → **Machine learning**; **Massively parallel and high-performance simulations**; • **Applied computing** → **Computational biology**.

Keywords

high-performance computing, machine learning, workflows, docking molecular dynamics, free energy estimation, COVID-19

ACM Reference Format:
Hyungro Lee, Andre Merzky, Li Tan, Mikhail Titov, Matteo Turilli, Dario Alfe, Agastya Bhati, Alex Brace, Austin Clyde, Peter Coveney, Heng Ma, Arvind Ramanathan, Rick Stevens, Anda Trifan, Hubertus Van Dam, Shunzhou Wan, Sean Wilkinson, and Shantenu Jha. 2021. Scalable HPC & AI Infrastructure for COVID-19 Therapeutics. In *Platform for Advanced Scientific Computing Conference (PASC '21), July 5–9, 2021, Geneva, Switzerland.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3468267.3470573

1 Introduction

Considering that there are about 10^{68} possible compounds to traverse for discovering effective drugs, efficient and effective frameworks for early-stage drug discovery [1] are required. In spite of the impact and potential of existing *in silico* methods, there is a need for a computationally efficient and better selection of candidates [2] that can proceed to later stages of the drug discovery protocol.

A fundamental challenge is a scale-accuracy trade-off. The search for novel drug candidates requires screening diverse and vast regions of the chemical space, orders of magnitude beyond those typically explored [3]. Yet, the exploration must be specific enough to select interesting candidates while keeping computational costs tractable. No single method is likely to achieve the necessary accuracy at the required scale. Thus, we need multiple methodological innovations that accelerate lead compound selections, utilizing advanced and scalable computational infrastructures.

The primary objective of this paper is to describe the computational infrastructure developed to support a novel computational campaign for *in silico* drug discovery. The campaign is comprised of multiple innovative methods realized as four workflows; the infrastructure enables those methods to run concurrently and as integrated workflows, while being scalable across diverse and heterogeneous high-performance computing (HPC) platforms. We examine the computational characteristics of those workflows and their implementations using the RADICAL-Cybertools middleware building blocks [4]. We discuss the computational capabilities required, the sustained and high-watermark performance, and the scale at which we execute those workflows.

This work is significant as it provides a scalable infrastructure for diverse workflows that are heterogeneous in distinct ways. The infrastructure is developed using a common set of middleware building blocks, delivers high-performance independent of the workflow and heterogeneity types, and does not optimize the performance of the workflow at the expense of others. It supports the flexible composition of workflows — each of which is a realization of a sophisticated and innovative method — into an integrated campaign with high end-to-end throughput. The infrastructure has supported a campaign that has delivered high-quality drug candidates utilizing $2.5{\times}10^6$ node-hours on diverse HPC platforms for: (i) docking $\sim 10^{11}$ ligands with a peak docking rate of $\sim 36{\times}10^6$ docks/hr — similar to the highest reported values [5]; (ii) AI-driven enhanced sampling simulations, which demonstrate 10× scientific improvement over traditional methods; and (iii) computing binding free energies on $\sim 10^5$ ligand-protein complexes, including 10^4 concurrently.

Our methods and infrastructure provide the computational fabric of the US-DOE National Virtual Biotechnology Laboratory in combination with resources from the EU Centre of Excellence in Computational Biomedicine. These methods and infrastructure have enabled the screening of more than 4.2 billion molecules against over a dozen drug targets in SARS-CoV-2. So far, over 1000 compounds have been identified and experimentally validated, resulting in advanced testing for dozens of hits. The campaign used diverse HPC platforms, including TACC Frontera, LLNL Lassen, ANL Theta, LRZ SuperMUC-NG and ORNL Summit.

This paper is organized as follows: In §2 we outline the scientific methods, their computational properties, and the computational campaign to coordinate them. In §3 we discuss the middleware building blocks used to develop infrastructure. We describe how our design and implementation support the execution of the diverse workflows of our campaign on heterogeneous HPC platforms and at scale. §4 characterizes the performance of our workflow executions, highlighting how we overcome challenges of scale and system-software fragility. §5 presents diverse measures of scientific impact towards therapeutic advances, discussing representative science results emanating from using these methods that required the coupling of multiple computational stages. §6 discusses related work, and §7 closes with a discussion of the impact, implications, and future trends.

2 Computational Campaign

The goal of our campaign is to discover new 'hits' — drug-like small molecules — and optimize those hits to viable 'lead' molecules that show potential to inhibit the viral activity. The campaign consists of two iterative loops (Fig. 1): the Hit to Lead Loop and the Lead Optimization Loop. In the first loop, ensemble docking programs (§2.1) 'dock' small molecules against ensembles of conformational states determined from a particular COVID-19 protein target. The results are used to train a docking surrogate model, followed by a coarse-grained (CG), enhanced sampling of molecular-dynamics with approximation of continuum solvent (ESMACS-CG) [6] free-energy estimation to determine whether the docked molecule and the protein target of interest can indeed interact (§2.4). Outputs from ESMACS–CG are input into ML-driven enhanced sampling methods (§2.3), which are taken through a further fine-grained (FG) refinement of the binding free-energy (ESMACS-FG; §2.4). Simultaneously, a secondary iterative loop is developed for a subset of compounds that show promising results from ESMACS–CG , where certain functional groups of promising molecules are optimized for protein-target interactions, using thermodynamic integration with enhanced sampling (TIES) [7] — a method of lead optimization (§ 2.5). We now describe these methods before discussing their implementation into workflows (WF1-4) in §4 and the software infrastructure to execute them §3.

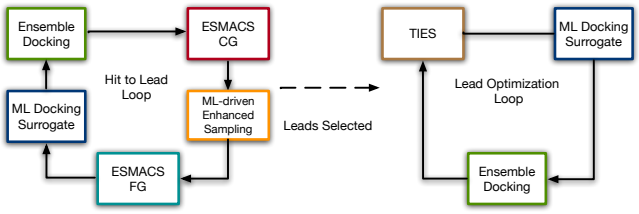

Figure 1: The computational campaign to advance COVID-19 therapeutics has two coupled loops: drug candidates go through four stages in the Hit-to-Lead loop; a small set of drugs are selected for the Lead Optimization loop. The following methods and protocols are implemented as distinct workflows (WF): Ensemble Docking (WF1), ML-driven Enhanced Sampling (WF2), both coarse-grained (CG) and fine-grained (FG) ESMACS (WF3), and TIES (WF4).

2.1 Ensemble Docking

Protein-ligand docking is a computational pipeline of ligand 3D structure (conformer) enumeration, exhaustive docking and scoring, and pose scoring. The input requires a protein structure with a designed binding region or a crystallized ligand from which a region can be inferred, as well as a database of molecules to dock in SMILES format — a compact representation of a 2D molecule.

Taking the 2D structures to 3D structures ready for structural docking, proteinization, and conformer generation is performed using Omega–Tautomers and, if stereochemistry is not specified, enantiomers are enumerated prior to conformer generation [8]. Typically, tautomers and enantiomers are enumerated for the incoming proposed analog or perturbation to the previous ligand. Conformer generation is performed on the ensemble of structures, generating 200-800 3D conformers for every enantiomer and reasonable tautomer generated. Once the set of 3D structures are enumerated from the 2D smiles, each one is docked against the pocket and scored. The best scoring pose is returned along with its ChemGauss4 score from exhaustive rigid docking [9].

2.2 ML Docking Surrogate Models

Scoring functions are used to score poses in order to determine the most likely pose of the molecule, the magnitude of which is used to provide an indication of active versus inactive ligands, and to rank order sets of libraries. We create an ML surrogate model to replace the use of docking as a means of locating regions of chemical space likely to include strong binding drug leads. The only free variable for the surrogate ML ranking function is the basic molecular information, which typically presents as a SMILES string. We use a simple featurization method, namely 2D image depictions, as they do not require complicated architectures such as graph convolution networks while demonstrating good prediction. We obtain these image depictions from the nCov-Group Data Repository [10], which contains various descriptors for 4.2B molecules.

2.3 ML-Driven Enhanced Sampling

Machine learning tools: (i) quantify statistical insights into the time-dependent structural changes a biomolecule undergoes in simulations [11]; (ii) identify events that characterize large-scale conformational changes at multiple timescales; (iii) build low-dimensional representations of simulation data capturing biophysical or biochemical information; (iv) use these low-dimensional representations to infer kinetically and energetically coherent conformational sub-states; and (v) obtain quantitative comparisons with experiments.

Recently, we developed convolutional variational autoencoders (CVAE) that automatically reduce the high dimensionality of MD trajectories and cluster conformations into a small number of conformational states that share similar structural and energetic characteristics [12]. We apply these approaches to ESMACS and TIES simulations. We also used CVAE to drive adaptive simulations for protein folding, and demonstrated that adaptive sampling techniques could provide at least an order of magnitude speedup [13]. These approaches provide previously unavailable acceleration of "rare" events — important to study protein-ligand interactions while leveraging supercomputing platforms [13].

2.4 ESMACS (Hit-to-Lead Loop)

Hit-to-Lead (H2L) is a step in the drug discovery process where promising lead compounds are identified from initial hits generated at preceding stages. It involves evaluation of initial hits followed by some optimization of potentially good compounds to achieve nanomolar affinities. The change in free energy between free and bound states of protein and ligand, also known as binding affinity, is a promising measure of the binding potency of a molecule and is used as a parameter for evaluating and optimizing hits at the H2L stage.

The ESMACS [6] protocol for estimating binding affinities of protein-ligand complexes is employed. It involves performing an ensemble of molecular dynamics (MD) simulations followed by free energy estimation on the conformations using a semi-empirical method called molecular mechanics Poisson-Boltzmann Surface Area (MMPBSA). The free energies for the ensemble of conformations are analyzed in a statistically robust manner yielding precise free energy predictions for any given complex. This is particularly important given that the usual practice of performing MMPBSA calculations on conformations generated using a single MD simulation does not give reliable binding affinities. Consequently, ESMACS predictions can be used to rank a large number of hits based on their binding affinities. ESMACS is able to handle large variations in ligand structures and hence is very suitable for the H2L stage where hits have been picked out after covering a substantial region of chemical space. The information and data generated with ESMACS can also be used to train an ML algorithm to improve its predictive capability.

2.5 TIES (Lead Optimization Loop)

Lead Optimization (LO) is the final step of the pre-clinical drug discovery process. It involves altering the structures of selected lead compounds in order to improve their properties, such as selectivity, potency, and pharmacokinetic parameters. Binding affinity is a useful parameter to make *in silico* predictions about the effects of any chemical alteration in a lead molecule. However, LO requires theoretically more accurate (without much/any approximations) methods for predictions with high confidence. In addition, the relative binding affinity of pairs of compounds that are structurally similar is of interest, rendering ESMACS unsuitable for LO. We employ thermodynamic integration with enhanced sampling (TIES) [7], which is based on an alchemical free energy method called thermodynamic integration (TI) [14], which fulfills conditions for LO. Alchemical methods involve calculating free energy along a non-physical thermodynamic pathway to get relative free energy between the two end-points. Usually, the alchemical pathway corresponds to the transformation of one chemical species into another defined with a coupling parameter (λ), ranging between 0 and 1. TIES involves performing an ensemble simulation at each λ value to generate the ensemble of conformations to be used for calculating relative free energy. It also involves performing a robust error analysis to yield relative binding affinities with statistically meaningful error bars. The parameters such as the size of the ensemble and the length of simulations are determined, keeping in mind the desired level of precision in the results [7].

3 Software Infrastructure

The computational campaign comprised of methods described in §2 are realized as diverse workflows of heterogeneous tasks. They require scalable software infrastructure that supports the concurrent execution of distinct workflows. The infrastructure must deliver needed performance that is invariant of workflow properties and type of heterogeneity. Currently, there are no "turnkey solutions" to support such campaigns across multiple heterogeneous HPC platforms, with the necessary performance and scale to ensure the required throughput. We leveraged RADICAL-Cybertools (RCT) [4] to develop middleware capable of supporting the campaign of four workflows that embody the methods and protocols illustrated in §2.

We briefly introduce RCT, outlining their novel capabilities. Note that we use the term "task" to indicate a stand-alone process that has well-defined input, output, termination criteria, and dedicated resources. For example, a task can indicate an executable that performs a simulation or a data processing analysis, executing on one or more compute nodes of an HPC platform. A workflow is comprised of tasks with dependencies, whereas a workload represents a set of tasks without dependencies or whose dependencies have been resolved. Thus, the tasks of a workload could, resources permitting, be executed concurrently.

3.1 RADICAL-Cybertools Overview

RCT are software systems developed to support the execution of heterogeneous workflows and workloads on HPC platforms. RCT have three main components: RADICAL-SAGA (RS) [15], RADICAL-Pilot (RP) [16, 17] and RADICAL-Ensemble Toolkit (EnTK) [18].

RS is a Python implementation of the Open Grid Forum SAGA standard GFD.90, a high-level interface to distributed infrastructure components like job schedulers, file transfer, and resource provisioning services. RS enables interoperability across heterogeneous distributed infrastructures.

RP is a Python implementation of the pilot paradigm and architectural pattern [19]. Pilot systems enable users to submit pilot jobs to computing infrastructures and then use the resources acquired by the pilot to execute one or more workloads. Tasks are executed concurrently and sequentially, depending on the available resources. RP can execute single or multi-core tasks within a single compute node or across multiple nodes. RP isolates the execution of each task into a dedicated process, enabling concurrent execution of heterogeneous tasks. RP's ability to concurrently execute heterogeneous tasks is unique compared to other pilots, and the distinctive feature motivating its use.

EnTK is a Python implementation of a workflow engine designed to support the programming of applications comprised of ensembles of tasks. EnTK executes tasks concurrently or sequentially, depending on their priority relation. Tasks are grouped into stages and stages into pipelines, depending on the priority relation among tasks. Tasks without reciprocal priority relations can be grouped into the same stage; tasks that need to be executed before other tasks have to be grouped into different stages. Stages are grouped into pipelines, and, in turn, multiple pipelines can be executed either concurrently or sequentially. EnTK uses RP, allowing the execution of workflows with heterogeneous tasks.

3.2 Supporting Multiple Task Execution Modes

Workflows 1–4 (based on methods outlined in §2) have different task types and performance requirements that, in turn, require different execution approaches. We discuss three pilot-based task execution frameworks developed to support the execution of workflows 1–4. We provide a brief comparison of the three approaches.

3.2.1 Execution Mode I: RAPTOR RP executes a special type of task, called "worker", that interprets and executes Python functions. We used this feature to implement an RP-based master/worker framework called RAPTOR (RAdical-Pilot Task OveRlay) to distribute multiple Python functions across multiple workers. RAPTOR enables parallel execution of those functions while RP implements capabilities to code both master and worker tasks and schedule their execution on the HPC resources acquired by submitting a job.

Fig. 2a illustrates the implementation of RAPTOR on Summit. Once RP has acquired its resources by submitting a job to Summit's batch system, RP bootstraps its Agent (Fig. 2a-1) and launches a task scheduler and a task executor (Fig. 2a-2). RP Scheduler and RP Executor schedule and launch one or more masters on one of the compute nodes (Fig. 2a-3) via either JSRUN [20] or PRRTE [21]. Once running, each master schedules one or more workers on RP Scheduler (Fig. 2a-4). Those workers are then launched on more compute nodes by RP Executor (Fig. 2a-5). Finally, each master schedules function calls on the available workers for execution (Fig. 2a-6), load-balancing across workers to obtain maximal resource utilization. The only change needed to use RAPTOR on diverse HPC platforms is a switch of the launch method for the master and worker tasks, e.g., on Frontera, from JSRUN to srun.

3.2.2 Execution Mode II: Using multi-DVM RP supports diverse task launch methods, depending on the availability of specific software systems on the target resources. On ORNL Summit, TACC Frontera, and LLNL Lassen, RP supports the use of the Process Management Interface for Exascale (PMIx) and the PMIx Reference RunTime Environment (PRRTE) [21]. PMIx is an open-source standard that provides methods to interact with system-level resource managers and process launch mechanisms. PRRTE provides a portable runtime layer that users can leverage to launch a PMIx server. PRRTE includes a persistent mode called Distributed Virtual Machine (DVM), which uses system-native launch mechanisms to bootstrap an overlay runtime environment, which can be used to launch tasks via the PMIx interface.

Using PRRTE/PMIx to place and launch stand-alone tasks on thousands of compute nodes allows for multiple concurrent DVMs. This enables partitioning of the task execution over multiple, independent sub-systems, reducing the communication and coordination pressure on each sub-system. This improves performance and resilience to PRRTE/PMIx implementation fragility.

Fig. 2b shows the integration between RP and PRRTE/PMIx on Summit. RP bootstraps its Agent (Fig. 2b-1) and, different from the RAPTOR implementation described in Fig. 2a, the Agent launches a set of DVMs, each spanning multiple compute nodes (Fig. 2b-2). The Agent also uses `ssh` to execute one or more RP Executor on one or more compute nodes (Fig. 2b-3). Once the DVMs and executors become available, RP schedules tasks on each executor (Fig. 2b-4). Each executor then uses one or more DVMs to place and then launch those tasks (Fig. 2b-5).

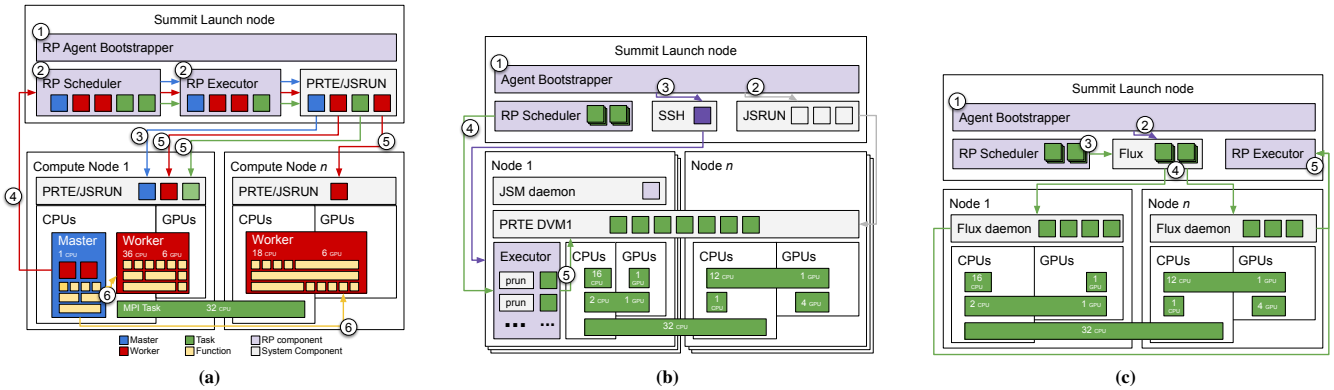

Figure 2: Pilot-based task execution frameworks implemented using RADIAL-Pilot. Numbers indicate the task execution processes. Blue box = RAPTOR master; red box = RAPTOR worker; green box = tasks; purple box = RP component; gray box = third-party software component. (a) RAPTOR's masters/workers executed via the standard RP capabilities. (b) RP executes heterogeneous tasks on one or more PRRTE/PMIx distributed virtual machines, each spanning multiple compute nodes. (c) Seamless integration of RP and Flux: RP schedules heterogeneous tasks that are placed and launched by Flux.

3.2.3 Execution Mode III: Flux PRRTE/PMIx introduce a variety of overheads [22]) and their current implementations are still fragile, especially when scheduling more than 20,000 tasks on more than 32 DVMs. Overheads and fragility lead to low resource utilization and unrecoverable failures. For these reasons, RP also supports the use of Flux [23] as an alternative system to place and launch tasks implemented as stand-alone processes. Fig. 2c illustrates the integration between RP and Flux. RP first bootstraps on one of the compute nodes of the HPC platform (Fig. 2c-1, Ref. [16]), and then launches Flux (Fig. 2c-2), scheduling tasks on it for execution (Fig. 2c-3). Flux places and launches tasks on Summit compute nodes via its daemons (Fig. 2c-4). RP Executor keeps track of task completion (Fig. 2c-5), and communicates this information to RP Scheduler, based upon which RP Scheduler passes more tasks to Flux for execution.

3.3 Enabling Heterogeneous Task Placement

Depending on the task launch method, RP places tasks on specific compute nodes, cores and GPU (Figs. 2a and 2b). This placement allows for efficient scheduling of tasks on heterogeneous resources. When scheduling tasks that require different amounts of cores and/or GPUs, RP keeps tracks of the available slots on each compute node of its pilot. Depending on availability, RP schedules MPI tasks within and across compute nodes.

Currently, RP supports four scheduling algorithms: continuous, torus, noop and flux. Continuous is a general purpose algorithm that enables task ordering, task colocation on the same or on different nodes, based on arrival order or explicit task tagging. Torus is a special-purpose algorithm written to support the BlueGene architectures; noop allows to pass single or bulk tasks keeping track only of their execution state; and flux delegates task placement and launching to the Flux framework.

RP opens a large optimization space for specific scheduling algorithms. For example, our continuous scheduler prioritizes tasks that require comparatively large amount of cores/GPUs so to maximize resource utilization. This could be further extended with explicit clustering or by including information about the execution time of each task.

We use RP capabilities to concurrently execute ESMACS and TIES methods (§2.4 and §2.5), reducing time-to-solution and improving resource utilization at scale. Both involve multiple stages of equilibration and MD simulations of protein-ligand complexes. Specifically, ESMACS protocol uses the OpenMM MD engine on GPUs, while the TIES protocol uses NAMD on CPUs. Leveraging RP's capabilities, we merge these two "workflows" into an integrated hybrid workflow with heterogeneous tasks which utilize CPU and GPU concurrently.

Fig.3 is a schematic where OpenMM simulations are tasks placed on GPUs, while NAMD simulations are MPI multi-core tasks placed on CPUs. Given that one compute node on Summit has 6 GPUs and 42 CPUs [24], we are able to run 6 OpenMM tasks in parallel which need 1 GPU and 1 CPU each. For optimal resource utilization, we assign the remaining 36 CPUs on one node to 1 NAMD task with 36 MPI ranks. NAMD tasks run concurrently on CPUs with the OpenMM tasks running on GPUs for *heterogeneous parallelism*. In order to achieve the optimal processor utilization, CPU and GPU computations must overlap as much as possible. We experimentally evaluate heterogeneous parallelism in §4.4.

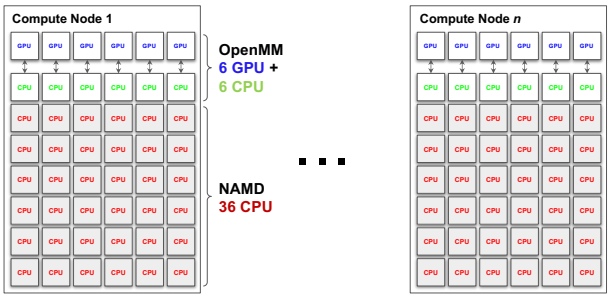

Figure 3: Using RP's heterogeneous task placement for hybrid workflow execution: RP concurrently executes OpenMM – which run on GPUs, and NAMD – which runs on CPUs, on the same Summit node(s) [24]. This increases node utilization significantly.

3.4 Coupling ML and HPC

To support ML-driven enhanced sampling (§2.3), we developed DeepDriveMD [13], to employ deep learning techniques, pre-trained models and tuned hyperparameters in conjunction with MD simulations for adaptive sampling. Specifically, DeepDriveMD couples a deep learning (DL) network — called convolutional variational autoencoder (CVAE)) — to multiple MD simulations, to cluster MD trajectories into a small number of conformational states. Insights gained from clustering is used to steer the ensemble of MD simulations. This may include either starting new simulations (i.e., expanding the pool of initial MD simulations), or killing unproductive MD simulations (i.e., simulations stuck in metastable states).

DeepDriveMD supports the following computational approach: (1) use an ensemble of MD simulations to generate initial MD data; (2) a 'training' run consisting of a ML algorithm; (3) an 'inference' step where novel starting points for MD are identified; and (4) spawn new MD simulations. DeepDriveMD is built upon EnTK, uses RP for advanced resource management, and is extensible to other learning methods and models, as well as other MD coupling schemes. The current implementation of DeepDriveMD utilizes Tensorflow/Keras (with Horovod for distributed data parallel training) and PyTorch. Typically, a run of DeepDriveMD requires 20 nodes on Summit.

4 Performance Characterization

We describe and analyze the performance of four workflows that represent the methods and protocols of § 2. We discuss the many application- and platform-level issues that need to be addressed to execute our pipeline at scale. We also quantify performance bottlenecks and their determinant factors, a necessary indication on how to further improve the pipeline for its deployment in sustained production. Overall, our performance analysis is an indicator of challenges and solution for pipelines that go beyond the specific requirements of our COVID-19 campaign.

4.1 WF1: Ensemble Consensus Docking

Compared to physics-based simulation methods, docking is a relatively inexpensive computational process. To increase the reliability of docking results, we prefer multiple docking protocols for the same ligand set and protease over individual docking scores. WF1 uses OpenEye and Autodock-GPU to leverage resource heterogeneity: the former executes on x86 architectures (e.g., Frontera); the latter on GPUs (e.g., Summit).

For each of the identified protein target[1] sites, WF1 iterates through a list of ligands and computes a docking score for that ligand-protein target pair. The score is written to disk and is used as filter to identify ligands with favorable docking results (score and pose). The docking call is executed as a Python function in OpenEye, and as a self-contained task process in AutoDock-GPU. In both cases, the RAPTOR framework (§3.2.1, Fig.2a) is used for orchestration.

The duration of the docking computation depends on the type of CPU (OpenEye) or GPU (AutoDock-GPU) used, and the computational requirements of each individual protein target. We measure the docking time (seconds) and docking rate (docks/hr) of three use cases: (1) production runs for NVBL-Medical Therapeutics campaigns; and runs for largest achievable size on (2) Frontera and (3) Summit. Table 1 summarizes the parameterization and results of the experiments we performed for each use case.

WF1 assigns one pilot for each protein target to which a set of ligands will be docked. Within each pilot, one master task is executed for every ≈100 nodes. Each master iterates at different offsets through the ligands database, using pre-computed data offsets for faster access, and generating the docking requests to be distributed to the worker tasks. Each worker runs on one node, executing docking requests across the CPU cores/GPUs of that node.

4.1.1 Use Case 1 We assigned each of the 31 targets to a single pilot, i.e., to an independent job submitted to the HPC machine's batch-queue. Due to the different batch-queue waiting times, at most 13 concurrent pilots executed concurrently. With 13-way pilot concurrency, the peak throughput was $\approx 17.4 \times 10^6$ docks/hr. To keep an acceptable load on Frontera's shared filesystem, only 34 of the 56 cores available were used.

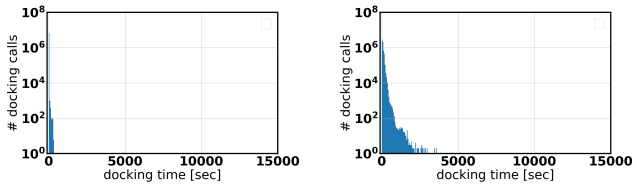

Figure 4: WF1, Use Case 1: Distribution of docking runtimes with the (a) shortest and (b) longest average docking time out of the 31 protein targets analyzed. The distributions of the docking runtimes all 31 protein targets have a long tail.

Figs. 4a and 4b show the distribution of docking times for protein targets with the shortest and longest average docking time, using the `Orderable-zinc-db-enaHLL` ligand database. All protein targets are characterized by long-tailed docking time distributions. Across the 31 protein targets, the min/max/mean docking times are 0.1/3582.6/28.8 seconds (Tab. 1), posing a challenge to scalability due to the communication and coordination overheads. The long tail distributions necessitate load balancing across available workers to maximize resource utilization and minimize overall execution time.

We addressed load balancing by: (i) communicating tasks in bulk so as to limit the communication frequency and therefore overhead; (ii) using multiple master processes to limit the number of workers served by each master, avoiding bottlenecks; (iii) using multiple concurrent pilots to partition the docking computations of the set of ligands.

Figs. 5a and 5b show the docking rates for the pilots depicted in Figs. 4a and 4b, respectively. As with dock time distributions, the docking rate behavior is similar across protein targets. It seems likely that rate fluctuations depend on the interplay of machine performance, pilot size, and specific properties of the ligands being docked, and the target protein target. We measure a min/max docking rate of $0.2/17.4 \times 10^6$ docks/hr with a mean of 5×10^6 docks/hr (Tab. 1).

[1] We define a protein target as a specific PDB file with a well defined binding site (according to how the specific molecular docking code requires) against which we dock the small molecule libraries.

Table 1: WF1 use cases. For each use case, RAPTOR uses one pilot for each protein target, computing the docking score of a variable number of ligands to that protein target. OpenEye and AutoDock-GPU implement different docking algorithms and docking scores, resulting in different docking times and rates. Resource utilization is often impeded by the long tail docking time distributions which cause an expensive cooldown period. However, the steady state resource utilization is >=90% for all use cases.

Use Case	Platform	Application	Nodes	Pilots	Ligands [$\times 10^6$]	Utilization avg/steady	Docking Time [sec] min	Docking Time [sec] max	Docking Time [sec] mean	Docking Rate [$\times 10^6$/hr] min	Docking Rate [$\times 10^6$/hr] max	Docking Rate [$\times 10^6$/hr] mean
1	Frontera	OpenEye	128	31	205	90% / 93%	0.1	3582.6	28.8	0.2	17.4	5.0
2	Frontera	OpenEye	7650	1	126	79% / 95%	0.1	14958.8	61.5	20.1	35.8	25.2
3	Summit	AutoDock-GPU	1000	1	57	95% / 95%	0.1	263.9	36.2	10.9	11.3	11.1

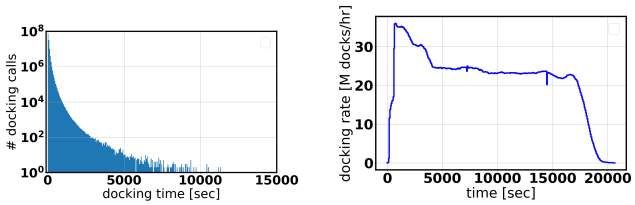

Figure 6: WF1, Use Case 2: (a) Distribution of docking time and (b) docking rate for a single protein target and 126×10^6 ligands. Executed with 158 masters, each using ≈50 compute nodes/2800 cores on Frontera.

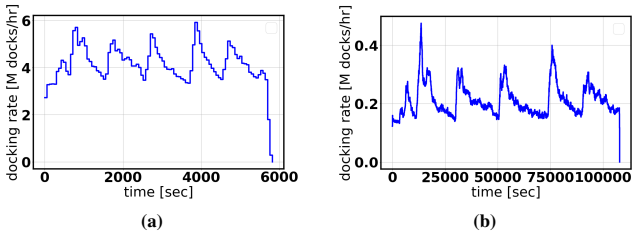

Figure 5: WF1, Use Case 1: Docking rates for the protein target with (a) shortest and (b) longest average docking time.

4.1.2 Use Case 2 Fig. 6a shows the distribution of docking times of approximately 126×10^6 ligands from the `mcule-ultimate-200204-VJL` library to a single protein target using OpenEye on Frontera. Note that the distribution is highly dependent on the protein target being used: for the specific protein target used in this run, we measure a min/max of 0.1/14985.8 seconds and a mean of 61.5 seconds (Tab. 1). The set of protein targets available to us varied in mean docking time from ≈3 to ≈70 seconds.

Fig. 6b shows the docking rate for a single pilot with 7650 compute (428,400 cores at 56 cores/node). Compared to Use Case 1, the rate does not fluctuate over time. After peaking at $\approx 35.8 \times 10^6$ docks/hr, the rate stabilizes at $\approx 25 \times 10^6$ docks/hr until the end of the execution (Tab. 1). Note that the long tail distribution of runtimes results in a long tail of docking calls and thus on a long "cooldown" phase. That phase ultimately lowers utilization from 92.3% in the steady-state (before cooldown starts) to a total average of 79.3%.

As discussed, the docking times depend on the protein targets used, and thus the docking rate inversely depends on that protein target choice. The range of rates is very wide: for the protein targets

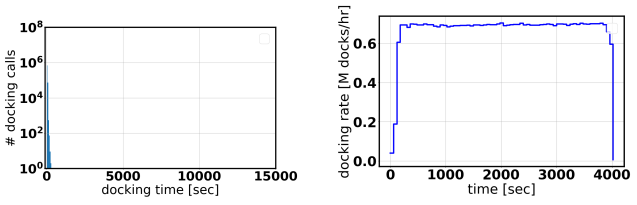

Figure 7: WF1, Use Case 3: (a) Distribution of docking time and (b) docking rate for a single protein target and 57×10^6 ligands. A pilot is concurrently executed on Summit with 6000 GPUs.

available to us, we observed a mean docking rate between $\approx 14 \times 10^6$ and $\approx 300 \times 10^6$.

4.1.3 Use Case 3 Figure 7a shows the distribution of the docking times of $\approx 57 \times 10^6$ ligands from the `mcule-ultimate-200-204-VJL` database to a single protein target using AutoDock-GPU on Summit. The distribution has a min/max/mean of 0.1/263.9/36.2 seconds (Tab. 1). Compared to Use Case 1, Fig. 4, max docking time is shorter, but the mean is longer. Compared to Use Case 2, Fig. 6a, both max and mean are shorter. As observed, those differences are due to specific properties of the docked ligands and the target protein target.

Fig. 7b shows the docking rate for a single pilot with 1000 compute nodes, i.e., 6000 GPUs. Different from Use Case 1 and 2, the rate peaks very rapidly at $\approx 11 \times 10^6$ docks/hr and maintains that steady rate until the end of the execution. The cooldown phase is also very rapid. We do not have enough data to explain the observed sustained dock rate. As with Use Case 2, we assume an interplay between the scoring function and its implementation in AutoDock-GPU and specific features of the 57×10^6 docked ligands.

Different from OpenEye on Frontera, AutoDock-GPU bundles 16 ligands into one GPU computation in order to efficiently use the GPU memory, reaching an average docking rate of 11.1×10^6 docks/hr (Tab. 1). Currently, our profiling capabilities allow us to measure GPU utilization with 5% relative error. Based on our profiling, we utilized between 93 and 98% of the available GPU resources.

4.2 WF2: ML-Driven Enhanced Sampling

WF2 is an iterative pipeline composed of 4 stages. After the first iteration of the 4 stages is completed, if outliers were found, the next iteration starts simulating those outliers; otherwise, the simulation continues from where it stopped in the previous iteration. The pipeline stops after a predefined number of iterations.

We measured RCT overhead and resource utilization of WF2 to identify performance bottlenecks. We define RCT overhead as the time spent not executing at least one task. For example, the overhead includes the time spent bootstrapping environments before tasks execution, communicating between EnTK and RabbitMQ (RMQ), or between EnTK and RP while workloads wait to execute. Resource utilization is the percentage of time when resources (CPUs and GPUs) are busy running tasks.

The blue bars in Fig. 8 show RCT overheads for the first version of WF2 and how RCT overheads grew with iterations. WF2 may require a variable number of iterations. Thus, our goal was to reduce RCT overhead, and importantly, make it invariant of the number of iterations.

Initial analysis suggested multiple optimizations of WF2: some of these involved improving the deep learning model, the outlier detection, and RCT. For the latter, we improved the communication protocol between EnTK and RMQ, and we reduced the communication latency between EnTK and RMQ. We avoided sharing connections to RMQ among EnTK threads, reducing multiple concurrent connections, and reused communication channels whenever possible.

Fig. 8 (orange) shows the combined effects of improving DeepDriveMD and EnTK communication protocol which reduced the overheads by 57% compared to Fig. 8 (blue). However, they were still growing with the number of iterations. We moved our RMQ server to Slate, a container orchestration service offered by OLCF, which reduced the communication latency between EnTK and RMQ, as shown in Fig. 8 (green). The optimization allowed RCT overheads to be invariant up to 8 WF2 iterations.

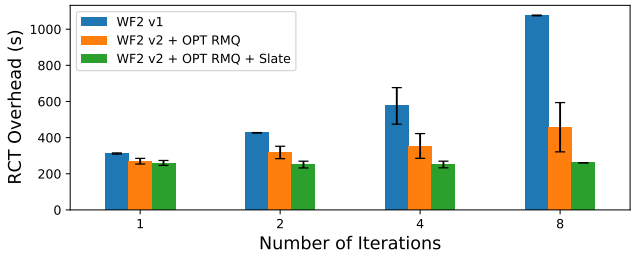

Figure 8: RCT overhead reduction with improved WF2, EnTK and RabbitMQ.

Fig. 9 depicts resource utilization for different (internal) RCT states as a time series. The region with "yellow, light blue, or green" colors represents unused resources; "dark" represents resource usage. Fig. 9 shows resource utilization of WF2, when executing four pipeline iterations on Summit with 20, 40, and 80 compute nodes. Note that most of the unused resources are CPU cores that are not needed by WF2. Overall, we measured 91%, 91%, 89% GPU utilization respectively. Across scales, Fig. 9 shows differences in the execution time of some of the pipeline stages but no relevant increase of the time spent without executing at least one task.

4.3 Hybrid WF3 & 4 Workflow

WF3 and WF4 are computationally intensive methods that cost several orders of magnitude more node-hours per ligand than WF1 [25]. As discussed in §3.3, WF3 and WF4 both compute binding free energies but have workloads comprised of distinct tasks: GPU-based

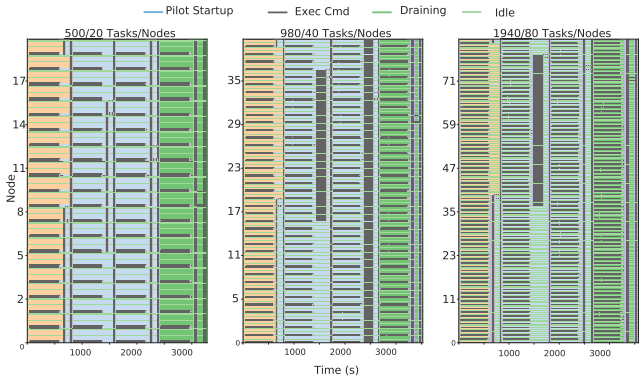

Figure 9: Timeline of RCT resource usage for WF2 when investigating weak scaling properties (from 20 to 80 nodes).

OpenMM and CPU-based NAMD tasks, respectively. Merging WF3 and WF4 into a single hybrid workflow allowed us to improve resource utilization by employing RP's unique capability to execute distinct tasks concurrently on CPU cores and GPUs. We evaluated that capability by measuring: (i) RCT overhead (as defined previously) as a function of scale; (ii) scalability as a function of problem and resource size; and (iii) resource utilization.

Fig. 10 compares RCT overhead to workflow time to completion (TTX) on 32 nodes for different task counts, representing different production workflow configurations. TTX in Fig. 10(c) illustrates concurrent execution of GPU and CPU tasks. The modest increase in TTX compared to Fig. 10(b) is likely due to interference from sharing resources across tasks (Fig. 3), and some scheduler inefficiency. A careful evaluation and optimization will form the basis of further investigation. Fig. 10(d) plots the TTX for the *Hybrid-LB* scenario when the number of WF3 and WF4 tasks are selected to ensure optimal resource utilization. The number of WF3 tasks completed in Fig. 10(d) is twice the number of WF3 tasks completed in Fig. 10(c), with no discernible increase in TTX.

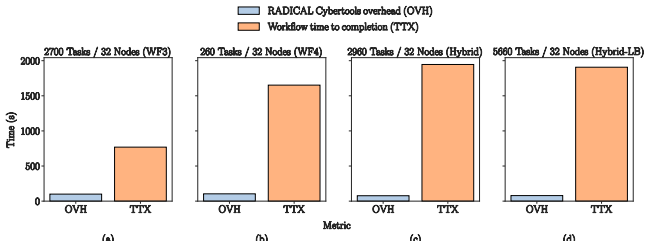

Figure 10: RCT Overhead in Hybrid Workflows.

Fig. 11 depicts RCT resource utilization for the configurations of Fig. 10(c) and Fig. 10(d). As with Fig. 9, "green space" represents unused resources; "dark space" represents resource usage. WF3 and WF4 have 4 and 3 stages, respectively, which can be discerned from black bars. Fig. 11(b) shows greater dark space and thus resource utilization than Fig. 11(a), representing the greater overlap of tasks on GPUs and CPUs due to workload sizing. Both have higher resource utilization than configurations in Fig. 10(a) and (b) due to concurrent CPU and GPU usage.

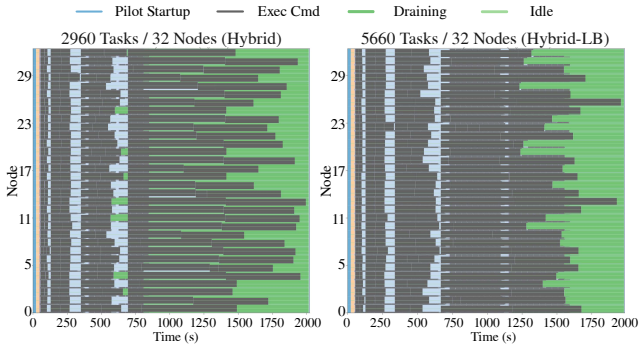

Figure 11: Timeline of RCT resource usage for Hybrid Workflows.

Fig. 12 shows the scalability of hybrid workflows with load balance enabled and up to 22640 tasks on 128 compute nodes on Summit. The left two panels show the comparison between 5660 GPU tasks and 5660 heterogeneous tasks (5400 GPU tasks + 260 CPU tasks). Note that RCT overhead is invariant between homogeneous and heterogeneous task placements and with proportionately increasing workloads and node counts (i.e., weak scaling).

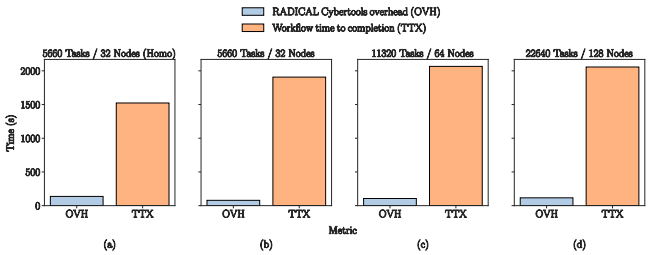

Figure 12: RCT Overhead in Hybrid Workflows at Scale.

In Figs. 10 and 12, RCT overhead varies from 3.8% to 11.5% of TTX but it should be noted that task runtimes for these experiments are significantly shorter than those of production runs. RCT overhead arises from state transitions and data movements and is essentially invariant of task runtimes, which are reproduced with fidelity in our experiments. Thus, in production runs, RCT overhead is a significantly less proportion of TTX.

4.4 WF3–4: Enhancing Scale and Reliability

Driven by science results (§ 5), and that WF3 & 4 are the "slowest" per ligand [25], we need to increase the number of nodes and improve reliability across multiple platforms. We preview results for WF3; experience with WF4 and hybrid WF3–4 execution will be reported subsequently.

We performed initial test runs using the multi-DVM execution mode described in §3.2.2, Fig. 2b and observed that executions were stable with each DVM running on < 50 nodes and executing < 200 tasks. Beyond that, we observed interruptions or connectivity losses between executors and DVMs. Further investigation will establish the causes of those limits and possible solutions for higher scalability.

We run WF3 on 1000 compute nodes (+1 node for RCT), executing 6000 1-GPU tasks on 32 concurrent DVMs. Each DVM spawned ≈32 nodes and executed up to 192 tasks. Fig. 13 shows the utilization of the available resources across different stages of the execution. The pilot startup time (blue) is longer than when using a single DVM [22], mainly due to the 336 seconds spent on launching DVMs which, currently, is a sequential process. Each task requires time to prepare the execution (purple), which mainly includes scheduling the task on a DVM, constructing the execution command, and process placement and launching by the DVM. The scheduling process takes longer than with a single DVM as it requires determining which DVM should be used. Further, constructing the execution command includes a 0.1s delay to let DVM finalize the previous task launching. As each operation is done sequentially per RP executor component, the 0.1s delay accounts for 600s alone.

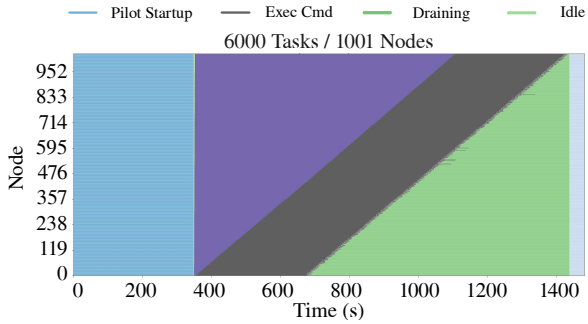

Figure 13: Timeline of RCT resource usage for WF3 using multi-DVM.

As with the other WF3–4 experiments, we reduced task runtimes to limit resource utilization while faithfully reproducing RCT overhead. In Fig. 13, Exec Cmd (task runtime) would be ten times longer for a production run. Thus, the overheads introduced by using multiple DVMs would have a lesser impact on the overall resource utilization.

We also run WF3 on 2000 compute nodes (+1 node for RCT), doubling the task and DVM number compared to the run with 1000 nodes. At that scale, we observed three main issues: (i) DVM startup failure; (ii) an internal failure of PRRTE; and (iii) lost DVM connectivity. The majority of tasks were successfully completed (11802 out of 12000), but those issues prevented RCT from handling their termination gracefully.

Given the current fragility of PRRTE/PMIx, we are investigating [17] executing WF3 with the RP/Flux integration described in §3.2.3, Fig. 2c. Tab. 2 summarizes the results of our initial test. Performance is comparable to RCT using a single-dvm, reducing the overheads measured with the multi-dvm implementation. We experienced no failures and are now working on deeper integration to scale our tests further. If the current results hold at higher scales, we plan to use the RP/Flux integration to run the WF3–4 pipeline in production on both Summit and Lassen.

5 Scientific Results

The previous section characterized the performance of the scalable infrastructure developed to support campaigns to advance COVID-19 therapeutics. Constituent workflows embody a diverse range of computational characteristics. Tab. 3 summarizes the heterogeneous platforms utilized and maps them to specific workflows supported.

Table 2: WF3 use case. Test runs with RP/Flux integration (§ 3.2.3, Fig. 2c).

Use Case	Platform	# Nodes	# Tasks	# Failed Tasks	Flux Resource Utilization	Task Scheduling Rate	Execution Time
WF3	Lassen	128	512	0	88%	14.21 t/s	6m

Table 3: HPC platforms used for the computational campaign. To manage the complexity arising from heterogeneity within and across platforms, requires middleware abstractions and design.

HPC Platform	Facility	Batch System	Node Architecture CPU	GPU	Workflows	Max # nodes utilized
Summit	OLCF	LSF	2 × POWER9 (22 cores)	6 × Tesla V100	WF1-4	2000
Lassen	LLNL	LSF	2 × POWER9 (22 cores)	4 × Tesla V100	WF2,3	128
Frontera	TACC	Slurm	2 × x86_64 (28 cores)	—	WF1	7650
Theta	ALCF	Cobalt	1 × x86_64 (64 cores)	—	WF1	256
SuperMUC-NG	LRZ	Slurm	2 × x86_64 (24 cores)	—	WF3-4	6000 (with failures)

Put together, the campaign has utilized $2.5{\times}10^6$ node-hours on these platforms to support: (i) docking $\sim 10^{11}$ ligands with a peak docking rate of up to $36{\times}10^6$ docks/hr; (ii) thousands of AI-driven enhanced sampling simulations; (iii) binding free energies on $\sim 10^5$ ligand-protein complexes, including 10^4 concurrently.

In addition to unprecedented scale and performance, the infrastructure provides unique qualitative insight which arises from the ability to integrate different methods. ML models are used to predict docking scores, i.e., the ranking of small molecules that potentially bind to and interact stably with the protein target of interest. ML models accurately rank-order a library of ligands in terms of predicted ranked score, using the regression enrichment surface (RES) technique to examine how well the ML models act as a surrogate for the scoring function [26]. The RES plot (Fig. 14 inset) shows the surrogate model efficiency for detecting true top-ranking molecules given a fixed allocation of predicted hits. For instance, if the computing budget allows n number of downstream simulations for inferred molecules of interest, the vertical line representing n on the x-axis of the plot shows the fraction of the real top-scoring compound distribution captured. Thus, the RES informs the number of top-scoring compounds needed to cover the chemical space of Mac1-specific molecules adequately.

Training this surrogate model required 100,000 randomly sampled docking scores from the data. Utilizing the remaining data for validation, we run the model to obtain predicted docking scores. For use in production workflows, it is important to know how many of the top predictions must we use to make sure a potentially exciting compound is not missed. A simple approach may guess if the model is decently accurate, then the top 50% or 10% might be needed. Since we have a validation set of data, we can calculate using the RES method this cutoff exactly [26]. We found model's predictive power allows two orders of magnitude efficiency over standard docking without surrogate model prefiltering of candidates. To calculate this, we ask what the minimal set of docking calculations must be performed to capture >99% of the top-scoring compounds. By prescreening with the surrogate model, the model's top 1% predictions capture nearly all of the highest-scoring 0.1% of docked compounds, thereby reducing the need to dock 99% of the original compounds.

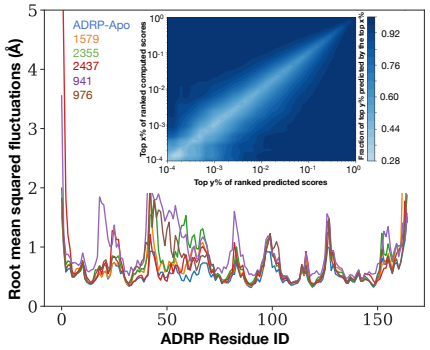

Figure 14: Root mean square fluctuation of Mac1 protein and 5 bound ligands. Different ligands induce different conformational changes in Mac1. The inset highlights the RES based on the ML-based surrogate model, showing a clear trend in improving the number of downstream calculations needed for finding n compounds in subsequent rounds of the iterative workflow.

While we could have filtered the top 0.1% of the predictions, reducing the docking load to a mere 0.1% of the dataset would only capture 50% of the highest scoring compounds, thus introducing some error. The infrastructure provided both the ability for large-scale ensemble docking, while using the surrogate to reduce the overall computational cost of the campaign.

We characterize the stability of compounds predicted to bind to the SARS-CoV-2 macrodomain (Mac1), which is part of the non-structural protein (nsp) 3. This enzyme is critical for the viral life-cycle by removing and recognizing host-derived ADP-ribosylation, a post-translational modification of host and pathogen proteins. Thus, Mac1 is an essential protein that can be targeted by small molecules such that its function can be inhibited and 'turn' the virus to be essentially non-pathogenic. We used DeepDriveMD (WF2) to study compounds that potentially interact with the primary binding site of Mac1[2]. To characterize the stability, we chose to examine the root mean square fluctuation (RMSF) analysis of the backbone C^{α}

[2]Although we studied over 200 compounds, we present results from the top five compounds that interact with Mac1 stably during O(100 ns) simulations

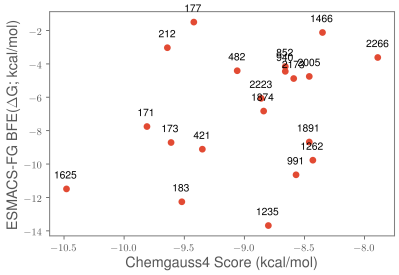

Figure 15: Correspondence between docking score (ChemGauss4/OpenEye) versus binding free energy using ESMACS-FG for Mac1. Although the correlation is low, ESMACS-FG acts as an effective *filter* for compounds with high affinity to Mac1 (e.g., compounds labeled 1625, 183, and 1235.)

atoms for the *apo*/ligand-free form of the Mac1 protein, which shows decreased fluctuations in the location of its backbone atoms compared to its *holo*/ligand-bound counterpart (Fig. 14). The ligand-bound protein undergoes higher RMSF fluctuations due to ligand-induced conformational changes, with notable displacements of a few residues such as Asp18-Val20, Asn54-Thr67, and His82. While these residues are not in direct contact with the ligand, the binding pocket structural changes causes a ripple effect on downstream residues in the protein.

We then used ESMACS (WF3) to predict which candidates bind tightly (Fig. 15). For instance, compound 1625 has a high affinity to Mac1 from both ESMACS as well as docking (indicated by the ChemGauss4 docking score and the ESMACS–FG binding free energy values) indicating favorable interactions. TIES (WF4) is used to refine such interactions between the protein and ligand: we performed TIES on 19 compound transformations, which entails mutating the original compound to new ligands with the goal of improving binding affinity. We used this method to study the effect of structural changes in a compound on their binding potency.

The relative binding affinities ($\Delta\Delta G$) predicted by TIES for these transformations fall between -0.55 to 4.62 kcal/mol. A positive value indicates a diminished relative binding potency for the "new" compound, whereas a negative value means that the transformation studied is favorable. Twelve out of the 19 transformations studied have $\Delta\Delta G > +1$, which suggests that they all correspond to unfavorable structural changes. The $\Delta\Delta G$ for the remaining 7 is statistically zero, which indicates that the corresponding structural modifications do not affect the binding. Both the favorable and unfavorable interactions provide insight into finding compounds with high affinity.

6 Related Work

In silico drug design presents intellectual challenges and is driven by clear imperatives of societal good and economic incentives. Recent methodological and infrastructural advances range from improved docking protocols [27, 28] to sophisticated multi-stage pipelines [29], and scalable infrastructure [30] for drug discovery pipelines.

Ref. [31] presents new methods and infrastructure focused on virtual screening. Brute-force traditional docking is unlikely to be fast or sophisticated enough for *in silico* drug design. Enhancing the ability of traditional docking protocols to sample larger chemical space is critical. The recent studies that combine docking with machine learning methods [27, 28] report up to 6000x increase in chemical space sampled [28] — without notable loss of favorably docked entities. To overcome the limitation that single docking protocols are not universally reliable, Ref. [29] introduces VirtualFlow —an open-source platform that supports several powerful docking programs on scalable infrastructure.

ML is used to improve computer-aided drug discovery. Ref. [32] integrates ML with binding free energy calculations. ML is also used to extend quantitative structure-activity relationships approaches. For example, AMPL [33] provides extensible pipelines that support the building and sharing of ML models, along with automating model training to improve key pharma-relevant parameters predictions. AMPL highlights the trend towards sophisticated and open-source software platforms that can be deployed on diverse suitable scalable infrastructure as per requirements.

While clusters and clouds are pervasive and scalable platforms for pharma industry, HPC and leadership platforms continue to provide important and powerful platforms [34] for academia and government. Ref. [35] have recently developed drug discovery pipelines on supercomputers that integrate multiple molecular modeling techniques — temperature replica-exchange advanced sampling techniques with docking protocols. Similarly, Ref. [36] used ensemble-docking approaches to overcome limitations for single docking protocols. Ref. [30] impressively used all ≈27500 GPUs on the Summit supercomputer for a sustained (average) molecular docking of ≈19000 compounds per second [5] using Autodock-GPU [37]. Over one billion compounds were docked [5] to two SARS-CoV-2 protein structures with full optimization of ligand position and 20 poses per docking, each in under 24 hours.

Spurred by, but not limited to COVID-19, the aforementioned recent publications reiterate the importance of both methodological and infrastructural enhancements. Consequently, our infrastructure is significant, as it: (i) provides scalable infrastructure for diverse workflows that are heterogeneous in distinct ways; (ii) delivers high-performance independent of the workflow and heterogeneity types, and similar to the highest values previously reported [5]; and (iii) is developed using a common set of middleware building blocks which supports the flexible composition of diverse methods – ensemble docking, ML surrogates for docking, ML-enhanced advanced sampling and binding free energies of differing granularity – into an integrated campaign with high end-to-end throughput.

7 Discussion

Multi-scale biophysics-based computational lead discovery is an important strategy for accelerated drug development. However, in its current formulation and practice, exploring drug compound libraries at the scale of billions of molecules is too inefficient, even on the largest supercomputers. This work shows a path towards improvement by: (i) ML components paired with, and trained from physics-based components; and (ii) building the HPC and AI infrastructure necessary to enable methodological advances [25]. In doing so, this work provides an enhanced drug discovery pipeline for COVID-19 — a societal and intellectual grand challenge.

The effectiveness and impact of the infrastructure are evidenced by its use to sustain a campaign on multiple heterogeneous platforms over months to generate valuable scientific insight (§5). This work is a harbinger of the evolving role of supercomputers, viz., increasingly

important generators of data for powerful ML models (e.g., WF1). In general, supercomputers will have to support campaigns with diverse components, viz., physics-based simulations, data generation and analysis, and ML/AI tasks. These individual workflows have different computational characteristics and performance challenges. They encompass high-throughput function calls, ensembles of MPI-based simulations, and AI-driven HPC simulations. There are no "turnkey solutions" to support such campaigns across multiple heterogeneous platforms, with the necessary performance and scale to ensure the required throughput. This has necessitated the design, development, and iterative improvement of infrastructure to advance therapeutics for COVID-19 and beyond.

Acknowledgements: Research was supported by the DOE Office of Science through the National Virtual Biotechnology Laboratory; as part of the CANDLE project by the ECP (17-SC-20-SC); UK MRC Medical Bioinformatics project (grant no. MR/L016311/1), UKCOMES (grant no. EP/L00030X/1); EU H2020 CompBioMed2 Centre of Excellence (grant no. 823712), and support from the UCL Provost. Access to SuperMUC-NG (LRZ) was made possible by a special COVID-19 allocation award from the Gauss Centre for Supercomputing in Germany. Anda Trifan acknowledges support from the United States Department of Energy through the Computational Sciences Graduate Fellowship (DOE CSGF) under grant number: DE-SC0019323. We acknowledge support and allocation from TACC and OLCF.

References

[1] Regine S Bohacek, Colin McMartin, and Wayne C Guida. "The art and practice of structure-based drug design: A molecular modeling perspective". In: *Medicinal research reviews* 16.1 (1996), pp. 3–50.

[2] Jiankun Lyu et al. "Ultra-large library docking for discovering new chemotypes". In: *Nature* 566.7743 (2019), pp. 224–229.

[3] Xiwen Jia et al. "Anthropogenic biases in chemical reaction data hinder exploratory inorganic synthesis". In: *Nature* 573.7773 (2019), pp. 251–255.

[4] Matteo Turilli et al. "Middleware building blocks for workflow systems". In: *Computing in Science & Engineering* 21.4 (2019), pp. 62–75.

[5] Jens Glaser, Josh V. Vermaas, David M. Rogers, et al. "High-Throughput Virtual Laboratory for Drug Discovery Using Massive Datasets". In: *International Journal of High-Performance Computing Applications (to appear)* (2020).

[6] Shunzhou Wan et al. "Rapid and Reliable Binding Affinity Prediction of Bromodomain Inhibitors: A Computational Study". In: *Journal of Chemical Theory and Computation* 13.2 (2017), pp. 784–795.

[7] Agastya P. Bhati et al. "Rapid, Accurate, Precise, and Reliable Relative Free Energy Prediction Using Ensemble Based Thermodynamic Integration". In: *Journal of Chemical Theory and Computation* 13.1 (2017). PMID: 27997169, pp. 210–222. DOI: 10.1021/acs.jctc.6b00979. eprint: https://doi.org/10.1021/acs.jctc.6b00979. URL: https://doi.org/10.1021/acs.jctc.6b00979.

[8] "OpenEye Toolkits 2019.Oct.2". Version 2019.Oct.2. In: *Open Eye Scientific* (2019). URL: http://www.eyesopen.com/.

[9] Mark R Mcgann et al. "Gaussian docking functions". In: *Biopolymers: Original Research on Biomolecules* 68.1 (2003), pp. 76–90.

[10] Yadu Babuji et al. "Targeting SARS-CoV-2 with AI-and HPC-enabled lead generation: A First Data release". In: *arXiv preprint arXiv:2006.02431* (2020).

[11] Gia G. Maisuradze, Adam Liwo, and Harold A. Scheraga. "Principal Component Analysis for Protein Folding Dynamics". In: *Journal of Molecular Biology* 385.1 (2009), pp. 312–329. ISSN: 0022-2836. DOI: https://doi.org/10.1016/j.jmb.2008.10.018. URL: http://www.sciencedirect.com/science/article/pii/S0022283608012886.

[12] Debsindhu Bhowmik et al. "Deep clustering of protein folding simulations". In: *BMC Bioinformatics* 19.18 (2018), p. 484. DOI: 10.1186/s12859-018-2507-5. URL: https://doi.org/10.1186/s12859-018-2507-5.

[13] Hyungro Lee et al. "DeepDriveMD: Deep-Learning Driven Adaptive Molecular Simulations for Protein Folding". In: *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*. IEEE. 2019, pp. 12–19. DOI: 10.1109/DLS49591.2019.00007. eprint: 1909.07817.

[14] T. P. Straatsma, H. J. C. Berendsen, and J. P. M. Postma. "Free energy of hydrophobic hydration: A molecular dynamics study of noble gases in water". In: *The Journal of Chemical Physics* 85.11 (1986), pp. 6720–6727. DOI: 10.1063/1.451846. eprint: https://doi.org/10.1063/1.451846. URL: https://doi.org/10.1063/1.451846.

[15] Andre Merzky, Ole Weidner, and Shantenu Jha. "SAGA: A standardized access layer to heterogeneous distributed computing infrastructure". In: *SoftwareX* 1 (2015), pp. 3–8.

[16] Andre Merzky et al. "Using pilot systems to execute many task workloads on supercomputers". In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2018, pp. 61–82.

[17] Andre Merzky et al. "Design and Performance Characterization of RADICAL-Pilot on Leadership-class Platforms". In: *arXiv preprint arXiv:2103.00091* (2021).

[18] Vivek Balasubramanian et al. "Harnessing the power of many: Extensible toolkit for scalable ensemble applications". In: *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2018, pp. 536–545.

[19] Matteo Turilli, Mark Santcroos, and Shantenu Jha. "A comprehensive perspective on pilot-job systems". In: *ACM Computing Surveys (CSUR)* 51.2 (2018), pp. 1–32.

[20] Dino Quintero et al. *IBM High-Performance Computing Insights with IBM Power System AC922 Clustered Solution*. IBM Redbooks, 2019.

[21] Ralph H. Castain et al. "PMIx: Process management for exascale environments". In: *Parallel Computing* 79 (2018), pp. 9–29. ISSN: 0167-8191. DOI: https://doi.org/10.1016/j.parco.2018.08.002. URL: http://www.sciencedirect.com/science/article/pii/S0167819118302424.

[22] Matteo Turilli et al. "Characterizing the Performance of Executing Many-tasks on Summit". In: *IPDRM Workshop, SC19* (2019). https://arxiv.org/abs/1909.03057.

[23] Dong H Ahn et al. "Flux: a next-generation resource management framework for large HPC centers". In: *2014 43rd International Conference on Parallel Processing Workshops*. IEEE. 2014, pp. 9–17.

[24] Oak Ridge Leadership Computing Facility. *Summit User Guide*. URL: https://docs.olcf.ornl.gov/systems/summit_user_guide.html.

[25] Aymen Al Saadi et al. "IMPECCABLE: Integrated Modeling PipelinE for COVID Cure by Assessing Better LEads". In: *arXiv preprint arXiv:2010.06574* (2020). https://arxiv.org/abs/2010.06574.

[26] Austin Clyde, Xiaotian Duan, and Rick Stevens. "Regression enrichment surfaces: a simple analysis technique for virtual drug screening models". In: *arXiv preprint arXiv:2006.01171* (2020).

[27] Zhirui Liao et al. "DeepDock: Enhancing Ligand-protein Interaction Prediction by a Combination of Ligand and Structure Information". In: *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE. 2019, pp. 311–317.

[28] Francesco Gentile et al. "Deep Docking: A Deep Learning Platform for Augmentation of Structure Based Drug Discovery". In: *ACS Central Science* (2020). DOI: https://doi.org/10.1021/acscentsci.0c00229.

[29] Christoph Gorgulla et al. "An open-source drug discovery platform enables ultra-large virtual screens". In: *Nature* 580.7805 (2020), pp. 663–668.

[30] Josh Vincent Vermaas et al. "Supercomputing Pipelines Search for Therapeutics Against COVID-19". In: *Computing in Science & Engineering* (2020). DOI: 10.1109/MCSE.2020.3036540.

[31] W Patrick Walters and Renxiao Wang. *New Trends in Virtual Screening*. 2020. DOI: https://doi.org/10.1021/acs.jcim.0c01009.

[32] Derek Jones et al. *Improved Protein-ligand Binding Affinity Prediction with Structure-Based Deep Fusion Inference*. 2020. arXiv: 2005.07704 [q-bio.BM].

[33] Amanda J Minnich et al. "AMPL: A Data-Driven Modeling Pipeline for Drug Discovery". In: *Journal of Chemical Information and Modeling* 60.4 (2020), pp. 1955–1968.

[34] Richard E Trager et al. "Docking optimization, variance and promiscuity for large-scale drug-like chemical space using high performance computing architectures". In: *Drug discovery today* 21.10 (2016), pp. 1672–1680.

[35] Micholas Smith and Jeremy C Smith. "Repurposing therapeutics for COVID-19: supercomputer-based docking to the SARS-CoV-2 viral spike protein and viral spike protein-human ACE2 interface". In: (2020). DOI: https://doi.org/10.26434/chemrxiv.11871402.v4.

[36] Atanu Acharya et al. "Supercomputer-based ensemble docking drug discovery pipeline with application to Covid-19". In: *ChemRxiv* (2020). DOI: 10.26434/chemrxiv.12725465.

[37] Scott LeGrand et al. "GPU-Accelerated Drug Discovery with Docking on the Summit Supercomputer: Porting, Optimization, and Application to COVID-19 Research". In: *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*. 2020, pp. 1–10.