DeepDriveMD: Deep-Learning Driven Adaptive Molecular Simulations for Protein Folding

Hyungro Lee[1*], Heng Ma[2*], Matteo Turilli[1*], Debsindhu Bhowmik[3],
Shantenu Jha[1,4**], Arvind Ramanathan[2**]
[1]*RADICAL, ECE, Rutgers University, Piscataway,NJ 08854, USA*
[2]*Argonne National Laboratory*
[3]*Oak Ridge National Laboratory*
[4]*Brookhaven National Laboratory, Upton, NY, USA*
[*]*Joint First Authors*
[**]*Senior Authors*

Abstract*—Simulations of biological macromolecules play an important role in understanding the physical basis of a number of complex processes such as protein folding. Even with increasing computational power and evolution of specialized architectures, the ability to simulate protein folding at atomistic scales still remains challenging. This stems from the dual aspects of high dimensionality of protein conformational landscapes, and the inability of atomistic molecular dynamics (MD) simulations to sufficiently sample these landscapes to observe folding events. Machine learning/deep learning (ML/DL) techniques, when combined with atomistic MD simulations offer the opportunity to potentially overcome these limitations by: (1) effectively reducing the dimensionality of MD simulations to automatically build latent representations that correspond to biophysically relevant reaction coordinates (RCs), and (2) driving MD simulations to automatically sample potentially novel conformational states based on these RCs. We examine how coupling DL approaches with MD simulations can fold small proteins effectively on supercomputers. In particular, we study the computational costs and effectiveness of scaling DL-coupled MD workflows by folding two prototypical systems, viz., Fs-peptide and the fast-folding variant of the villin head piece protein. We demonstrate that a DL driven MD workflow is able to effectively learn latent representations and drive adaptive simulations. Compared to traditional MD-based approaches, our approach achieves an effective performance gain in sampling the folded states by at least 2.3x. Our study provides a quantitative basis to understand how DL driven MD simulations, can lead to effective performance gains and reduced times to solution on supercomputing resources.

***Index Terms*—deep learning, machine learning, molecular dynamics, protein folding**

I. Introduction

Understanding the biophysical processes that control how a polypeptide folds into its three-dimensional native structure remains an outstanding question in molecular biology. Experimental studies, simulations and theory have continued to provide valuable insights into how proteins fold, especially in the context of small, and fast folding proteins (typical folding times of about several μs-ms). [1] It is generally accepted that proteins fold through a discrete number of intermediate states, where each state consists of partial folded components in terms of secondary structures [2]. Associated with these intermediate states are timescales that characterize their stability (either in terms of how long a secondary structure may persist or other physical/ structural attributes) are before the protein 'jumps' into other states finally reaching its folded state.

The inherent high dimensionality of protein folding trajectories (generated from simulations) makes it challenging to characterize: (i) metastable states – states that share similarity in structure/ conformation, and other biophysical properties of interest, and (ii) transition times – how stable these intermediate states. A number of clustering approaches have therefore been developed for obtaining insights into metastable states and characterizing transition times [3]–[6]. Such approaches build reduced dimensional (latent) representations from molecular dynamics (MD) data, typically using principal component analysis or independent component analysis techniques [7], [8].

Complementary to such approaches, we recently developed a deep convolution variational autoencoder (CVAE) [9], to automatically cluster protein folding trajectories into a small number of conformational states. Our approach was able to organize the conformational landscape based on key reaction coordinates for protein folding such as the fraction of native contacts and the root mean squared deviations (RMSD) to the native state. Further, our approach also allowed us to transfer these learned properties across independent simulations.

In addition to the aforementioned challenges, MD simulations tend to get 'stuck' within metastable states [6]; a variety of approaches have been developed to address this challenge. These techniques, collectively referred to as enhanced sampling methods, use: (1) a pre-determined set of low dimensional representations referred to as reaction coordinates or collective variables determined from MD simulations either biophysically determined *a priori* [10]–[12] (e.g., distances between key residues [12]), or by learning latent dimensional representations (e.g., described above) to adaptively sample and accelerate protein folding/ or other biophysical phenomena of interest, and/or (2) importance sampling techniques that enhance 'rare' events in the simulations.

Most enhanced sampling techniques [13], [14] involve generating an initial pool of MD data (either a single simulation or an ensemble of simulations), followed by intermittently stopping and steering MD simulations towards novel starting points [15]. We recently generalized the above workflow [16]

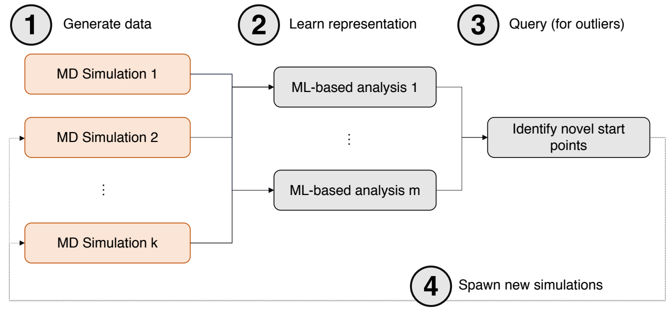

Fig. 1. Computational workflow ‘motif’ for coupling MD simulations with ML approaches.

to include learning driven MD simulations, which we now formalize into a computational motif (see Fig. 1) and implement at scale. The motif is characterized by: (1) generating an initial pool of MD data typically using a large ensemble of MD simulations, followed by (2) a ‘training’ run consisting of a ML algorithm, and (3) an ‘inference’ step where novel starting points for MD are identified and (4) new MD simulations are spawned. This may include either starting entirely new simulations (i.e., expanding the pool of initial MD simulations) or killing unproductive MD simulations (i.e., simulations that seem to be stuck in metastable states). We note that this computational motif represents many scientific discovery processes beyond protein folding.

In general, there are three primary motivations for the coupling of ML/DL driven approaches with traditional HPC simulations: (i) ML/DL can be used to reduce the computational cost of simulating a process via the creation of a computational surrogates; (ii) improve the effective performance of vanilla HPC simulations by using ML/DL driven HPC simulations, and (iii) use simulations to improve the training of ML/DL models (both in the presence of sparse data and otherwise). In this paper, we investigate the latter two scenarios and discuss software systems solution that provide generalized implementation and capabilities.

There are at least three different measures of performance of DL driven MD simulations of relevance: (A) Traditional measures of scale, scalability, and efficiency; (B) performance of a DL driven simulation method (or algorithm) relative to a non-DL driven “pure” simulation based method; and (C) the performance of the learning component as a function of the number of simulation elements coupled to the learning component. This paper discusses all three performance measures. However, it is difficult to assess performance measure C *a priori*, which in turn, influences the optimal partition between resources assigned to the ML/DL component and those assigned to HPC simulations. Performance measure C also influences performance measure B, and possibly also efficiency and scalability. Our software system has the ability to dynamically partition and balance resources assigned to the ML component and those assigned to HPC component to optimize performance measure C.

The contributions of this paper are three fold: (1) We design and implement DeepDriveMD — a framework for deep learning driven simulations using RADICAL-Cybertools. DeepDriveMD is not constrained to specific learning methods (e.g., CVAE) but can support arbitrary deep learning driven methods and HPC simulations; (2) We utilize DeepDriveMD for a VAE driven adaptive molecular simulations and show that it is possible to fold small proteins/peptides using on Summit — a leadership computing platforms at Oak Ridge National Laboratory; and (3) We provide performance measures for integrated learning-simulation methods, assessing the overall effectiveness of our workflow relative to non-DL driven simulations.

II. Related Work

Notable examples of implementation of the workflow motifs — partial and complete, include the REAP approach [13], where the authors define a mapping between a finite set of states and actions that enable an agent to achieve its goal based on a user defined set of order parameters (OPs) /reaction coordinates (RC). The weights on the RCs is initialized, with MD simulations used to learn which RCs contribute most to the final target. Similarly, Galvelis and Sugita defined an enhanced sampling protocol [17] that is based on nearest neighbor density estimator and a neural network to define a bias potential that resulted in ergodic sampling and characterizing free energy profiles for various polypeptides. Notably, this approach currently seems to be limited to 8-dimensional bias potentials. Wang, Ribeiro, and Tiwary use a VAE [18] similar to our approach, however, constrain the encoder/decoder with an information bottleneck that identifies an optimal RC. Other approaches such as the neural networks-based variationally enhanced sampling [19] and Boltzmann Generators [20] share similar workflow motifs, although the exact use of MD simulations versus other types of sampling (e.g., Markov chain Monte Carlo) may differ.

The approach investigated in this paper, and the other approaches described above are different from AlphaFold [21], where the target problem is to model the final folded 3-dimensional structure of a protein from its primary sequence.

The distinction between this work and aforementioned implementations of the motif are: (i) Methodological enhancement: this work investigates the interplay between simulations and learning. Specifically, it can adaptively tune the ratio of computational resources assigned to simulations and learning based upon Reconstruction loss; (ii) Scale: thanks to first-order middleware for HPC workflows, the scale of problem investigated is significantly greater than previously reported, and (iii) Generality: Our proposed motif and software system can support multiple learning methods. We demonstrate the impact using CVAE based DL method, but could just as well use other learning methods.

III. Computational Problem, Design and Implementation

The overall scientific goal of our paper can be summarized as follows: given an initial set of starting conformations, representing the unfolded state ensemble of a protein, run simulations to enable an efficient sampling of the final folded state using MD simulations. In the workflow, MD simulations are overseen by the CVAE model that collects the MD conformers as training input and in return identifies the state of each simulation for interative decision-making on whether to continue or terminate an individual MD task. Conformers in less populated latent space of the CVAE representation are selected as ‘outliers’ for instantiating a new MD task. The outliers are inferred on the basis of using the density based spatial clustering of applications with noise (DBSCAN) algorithm in the latent dimensions of the CVAE model with the lowest reconstruction loss [9]. Note that there are several choices for the selection of outliers; we used one that is known to work well in practice. The number of outliers identified is capped at 150 with a maximum of 10 members in each cluster (identified by DBCSAN). This is reasonable on the basis of the number of initial simulations carried out.

The workflow also requires setting up the MD and ML tasks, managed by a contemporary scheduler to administer the computational resource and enable interfacing between MD simulations and DL framework, on the specific architecture of Summit, an IBM AC922 system that integrates more than 27,000 NVIDIA V100 GPUs and 9,000 IBM Power9 CPUs. Note that on Summit, each node consists of 2 CPUs with 6 fully inter-connected GPUs using the NVLINK architecture. Despite the computational demanded fulfilled with Summit HPC system, the workflow also requires setting up the MD and ML tasks, managed by a contemporary scheduler to administer the computational resource and enable interfacing between MD simulations and DL framework on Summit. The MD task is carried out by GPU-accelerated OpenMM molecular simulation engine and VAE framework is set up with Keras/TensorFlow, also on GPU. Both tasks enable the workflow to fully leverage the GPU nodes on Summit. Here we adopt the RADICAL-Cybertools to conduct all the tasks of the workflow in a scalable fashion.

A. *RADICAL-Cybertools: Ensemble Execution on Summit*

The RADICAL-Cybertools (RCT) software stack is used to support the scalable concurrent and sequential execution of heterogeneous tasks on high-performance computing (HPC) resources. RCT are a set of software systems that serve as middleware to develop efficient and effective tools for scientific computing. Specifically, RCT enable executing ensemble-based applications at extreme scale [22] and on a variety of computing infrastructures.

RCT consists of three main components: RADICAL-Ensemble Toolkit (EnTK) [23], [24], RADICAL-Pilot (RP) [25], and RADICAL-SAGA(RS) [26], [27]. EnTK provides the ability to create and execute ensemble-based workflows/applications with diverse coordination and communication algorithms, abstracting the need for explicit resource management. EnTK uses RP as a pilot-based [28] runtime system to provide resource management and task execution capabilities. In turn, RP uses RS as an access layer towards HPC resources.

RCT adopts the “building blocks” approach to workflows [29]–[31]. RCT provide scalable implementations of building blocks in Python and are currently used to support dozens of scientific projects on HPC systems, including several existing and prior INCITE awards. RCT is increasingly being used to support applications that involve the concurrent and adaptive execution of ML and simulation tasks [32]. RCT has been used extensively to support biomolecular sciences algorithms/methods, e.g., replica-exchange, adaptive sampling and high-throughput binding affinity calculations.

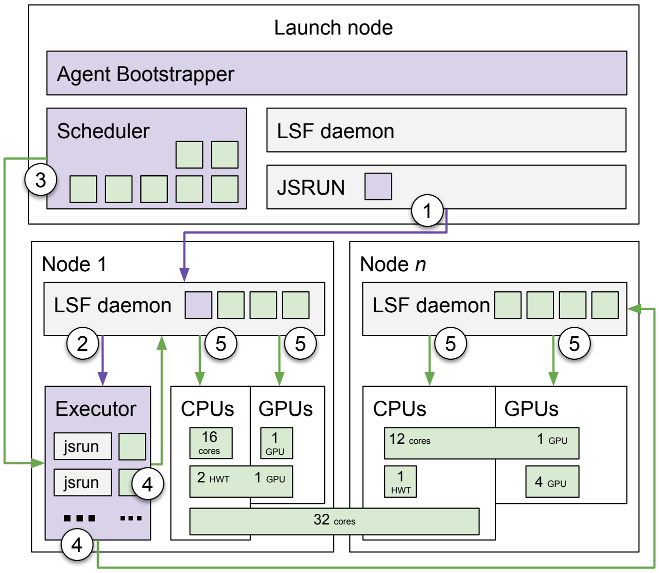

Fig. 2. RADICAL-Pilot (RP) deployment on Summit. Purple: RP components; Gray: IBM Platform Load Sharing Facility (LSF) components; Green: heterogeneous computational tasks. 1-2: scheduling of RP’s Executor component on a work node via LSF daemons and JSRUN; 3-5: scheduling of computational tasks via RP’s Scheduler component, LSF daemons and JSRUN. RP manages heterogeneous tasks that require arbitrary combinations of available resources.

RP is an implementation of the pilot abstraction, engineered to support scalable launching of heterogeneous tasks across different HPC platforms. RP is a runtime system designed to decouple resource acquisition from task execution. As every pilot system, RP acquires resources by submitting a batch job, then bootstraps dedicated software components on those resources to schedule, place and launch application tasks, independent from the machine batch system.

RP is a distributed system designed to instantiate its components across available resources, depending on the platform specifics. Each components can be individually configured so as to enable further tailoring while minimizing code refactoring. RP uses RS to support all the major batch systems, including Slurm, PBSPro, Torque and LSF. RP also supports many methods to perform node and core/GPU placement, process pinning and task launching like, for example, aprun, JSM, PRRTE, mpirun, mpiexec and ssh.

RP is composed of two main components: Client and Agent. Client executes on any machine while Agent bootstraps on one of Summit's batch nodes. Agent is launched by a batch job submitted to a batch system via RS. After bootstrapping, Agent pulls bundles of tasks from Client, manages the tasks' data dependencies if any, and then schedules tasks for execution via one or more launching methods. RP can execute scalar, OpenMP, MPI tasks within and across multiple nodes, allowing each task to use one or more CPU/GPU exclusively or concurrently.

RP has been ported to Summit enabling fine-grained mapping, scheduling and execution of heterogeneous computational tasks on CPU, GPU, and hardware threads (HWT). Agent deployment depends on several configurable parameters like, for example, number of sub-agents, number of schedulers and executors per sub-agent, and method of placing and launching tasks for each executor of each sub-agent. On Summit, the default deployment of Agent instantiates a single sub-agent, scheduler and executor on a batch node. The executor calls one `jsrun` command for each task, and each `jsrun` uses the JSMD demon to place and launch the task on work nodes resources (thread, core and GPU).

Fig. 2 shows an alternative deployment of Agent that uses PRRTE/DVM instead of JSM/LSF. to place and launch tasks across compute nodes. This configuration enables a sub-agent to use more resources than with JSM/LSF and improves scalability and performance of task execution. Note that, independent from the configuration and methods used, RP can concurrently place and launch different types of tasks that use different amount and types of resources. Our tests show reliable concurrent execution of up to 16384 tasks, each task using 1 core for a total of 404 compute nodes, and up to 100 tasks, each requiring 1096 cores.

EnTK exposes an application programming interface (API) for the description of scientific applications as static or dynamic sets or sequences of pipelines. Each pipeline is composed of stages and each stage contains an arbitrary set of tasks. Tasks can execute concurrently while stages can execute only sequentially. These properties are insured by design, offering what we have called a Pipeline Stage Task (PST) model for the specification of computational workflows. It is important to note that 'task' here are not functions, methods or sub-processes of one of EnTK components. Task indicates instead a self-contained process (i.e., program) executed and managed by the operating system of the target resource. Consistently, tasks can be a single-threaded, multi-threaded or MPI program, and can use CPUs, GPUs or both within and across the compute nodes of a target machine.

B. Integration of ML and MD

Many scientific workloads are comprised of many tasks, where each task is an independent simulation or data processing analysis. The execution of many tasks on heterogeneous HPC platforms requires scalable dynamic resource management and multi-level scheduling. Together, EnTK and RP enable the codification of many-task applications and their scalable execution on HPC machines like Summit.

In a recent paper [22], we characterized the performance of executing many tasks using RP when interfaced with JSM or PRRTE on Summit: RP is responsible for resource management and task scheduling on acquired resource; JSM or PRRTE enact the placement and launching of scheduled tasks. When using homogeneous single-core, 15 minutes-long tasks, PRRTE scales better than JSM for $> O(1000)$ tasks; PRRTE overheads are negligible; and PRRTE supports optimizations that lower the impact of overheads and enable resource utilization of 63% when executing O(16K) 1 core tasks over 404 compute nodes. In this paper, the workload is comprised of heterogeneous tasks of varying temporal durations but the resource utilization and scaling remain invariant.

For each experiment of this paper, we vary only the number of starting conformations, i.e., how many simulations are initiated across multiple GPUs on Summit. We explicitly choose only one Summit node, training our CVAE model on 4 out of the 6 GPUs available. This is a practical choice since the two peptides are small enough that they do not need additional compute resources for training our deep learning model. Similarly, once the training is complete, the same Summit node is also utilized for inference, i.e., to identify novel conformations determined by the CVAE.

IV. Results

A. Science use cases: Folding simulations of Fs-peptide

We considered two minimal use cases for our workflow. The first one consisted of folding simulations of the Fs-peptide (21 residues consisting of Ace-A_5(AAARA)$_3$A-NME, where Ace and NME represent the N- and C-terminal end caps of the peptide respectively, with A representing the amino acid Alanine and R representing Arginine) in *implicit* solvent conditions driven by a CVAE that learns latent representations from our simulations. Our simulations used the GBSA-OBC potentials and the AMBER-FF99SB-ILDN force-field set up similar to previous studies with an aggregate time of 18μs at 300 K. These simulations were set up in a similar way to previous studies (where each individual simulation was 500 ns); however, the length of any individual simulation in our work was limited to only 50 ns.

The second set of simulations consisted of a fast folding variant of the villin head piece (VHP; 35 amino acid residues) in *explicit* solvent simulations. These simulations used the AMBER-FF99SB-ILDN with the TIP3P water model, with a cubic box of $60 \times 60 \times 60$ $\mathring{A}^3$ dimensions. The simulations were carried out for an aggregate of 0.9μs at 300 K. Note that this timescale is limited by the wall time limits on Summit@OLCF as well as the use of explicit solvent simulations, which can take considerably longer wall clock time to simulate. Individual simulations were capped at 10 ns. In this example we noticed that as a consequence of rather limited sampling, our runs did not end up fully folding VHP to its native state. However, it does allow the simulations to sample partially folded states, where certain

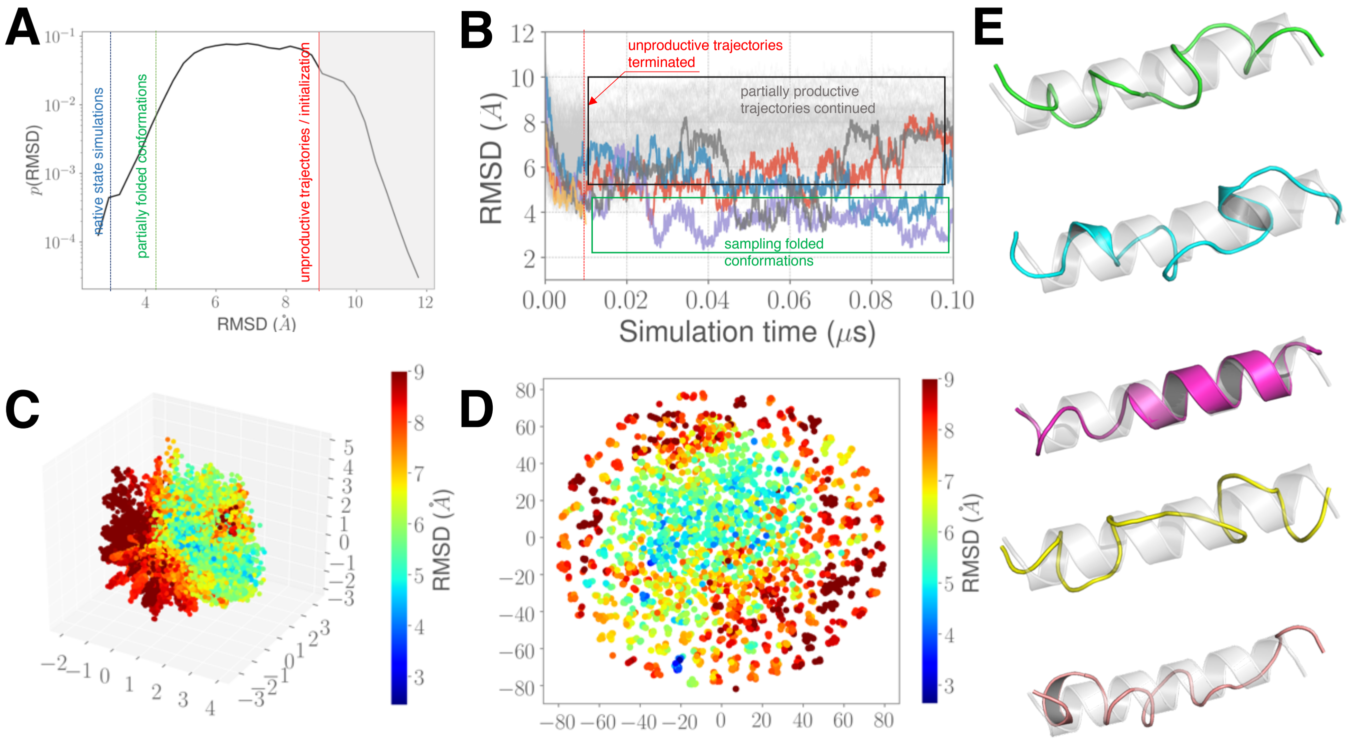

Fig. 3. Adaptive simulations of Fs-peptide folding: (A) A summary distribution of RMSD (see main text) to the native state of Fs-peptide from the adaptive simulations. Note the presence of a large number of unproductive trajectories that begin from unfolded state having a RMSD higher than 9.2$\mathring{A}$. (B) Time evolution of RMSD to the native state from the 720 simulations initiated on Summit GPUs using OpenMM simulation engine. The productive trajectories (i.e., trajectories that sample conformations with $< 4.3\mathring{A}$ RMSD to the native state at least once) are shown in different colors. (C) Summary of the CVAE learned representation shown using 3 dimensions (for visualization purposes; see main text) showing each conformation from the adaptive simulations as a 3D coordinate painted by its RMSD to the native state. Notably the states involving the unfolded ensemble cluster together, along with intermediate states also clustered together. (D) To delineate the folded ensemble, we project the CVAE learned presentation onto two dimensions using t-SNE where one can observe the separation between the folded and unfolded states. (E) Representative conformations from the successful trajectories (in panel B) are shown with respect to the native state. The lowest RMSD achieved is about 2.3$\mathring{A}$ shown in magenta. Other conformations sampled from the intermediate states are shown for completeness.

α-helical turns are formed. As our paper focuses largely in studying the computational performance and scaling aspects of deep learning approaches coupled to MD simulations, we do not present the results from our simulations.

Fig. 3 summarizes the results of using our workflow in simulating the folding process of Fs-peptide. We first evaluated the quality of folding observed from our simulations using the root-mean squared deviation (RMSD) with respect to the final folded state (a fully formed α-helix) from all of the simulations. As shown in Fig. 3A, the histogram presents a composite picture of the folding process where by a small proportion of the simulations seem to sample the folded α-helical states (labeled native state simulations). Further, a small number of simulations also sample partially folded states (RMSD cut-off of 4.3$\mathring{A}$). We also observe that a large portion of the simulations also sample fully unfolded states (RMSD cut-off $> 9.2\mathring{A}$). To further understand the time-evolution of the individual trajectories, we plotted the RMSD as a function of simulation time (Fig. 3B). One of the observations from the aggregate set of simulations is that all of the simulations begin with a high RMSD ($> 10\mathring{A}$ on average) and evolve gradually towards low RMSD values to the native state. Of the total 720 number of simulations initiated from the unfolded state, 5 of them sample partially folded states where as two of them sample close to the native state of the protein.

As posited in the beginning of our study, we used our CVAE to drive our enhanced sampling approach. We observed that building a latent representation consisting of 6 dimensions provided the best reconstruction of the simulation data. Since visualization in 6 dimensions is difficult, as shown in Fig. 3C, we selected 3 dimensions (from the 6) and used it to organize the conformational landscape. Each conformation in the plot is represented as a 3D coordinate and painted with the RMSD to the native state. Note that the RMSD is not part of our training data (only the contact matrices are used as input to train our CVAE) and is an emergent property from our analysis. Most of the unfolded conformations are localized to one region of our representation while many of the folded states are clustered together. We also used t-stochastic neighborhood embedding (t-SNE) to visualize the clustering in a 2D representation. Notably, the folded and unfolded states are separated out (red and blue dots). Additionally, representative structures

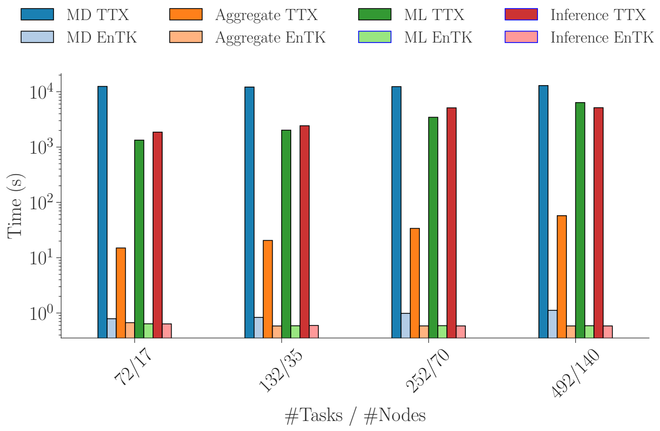

Fig. 4. Total time to execution (TTX) and EnTK time overhead of each stage of the workflow described in Fig.1.

extracted from the trajectories (from the five trajectories shown in different colors in Fig. 3B) showed the presence of various intermediates (Fig. 3E) in the folding process. Note that these states are only a representative subset of the conformations sampled from our adaptive sampling technique.

B. *Scaling Profiles for Adaptive Simulations*

RADICAL Ensemble Toolkit (EnTK) is designed to support the concurrent execution of computational pipelines. Each pipeline is composed of stages and each stage contains an arbitrary set of tasks. Tasks can execute concurrently while stages can execute only sequentially. These properties are insured by design, offering what we have called a Pipeline Stage Task (PST) model for the specification of computational workflows. It is important to note that 'task' here are not functions, methods or sub-processes of one of EnTK components. Task indicates instead a self-contained process (i.e., program) executed and managed by the operating system of the target resource. Consistently, tasks can be a single-threaded, multi-threaded or MPI program, and can use CPUs, GPUs or both within and across the compute nodes of a target machine.

Specified in PST, the workflow of Fig. 1 consists of a single pipeline with four stages. The first stage executes one or more MD simulations, the second stage aggregates the results, the third stage one or more ML training tasks on the aggregated data produced by the tasks of the first stage, and the fourth stage make an inference about the initial state of the next MD simulation. At this point, the workflow repeats until the protein folds.

EnTK offers two main benefits when implementing this workflow: the ability to arbitrarily change the number of tasks executed in each stage without significant programming or execution time overheads; and the ability to extend the workflow with as many MD simulations/ML inferences stages are required by the protein to fold. Here we focus on the first benefit, studying the relationship between the number of concurrent simulations executed, the amount of data generated, the number of simulated frames and the quality of the leaning we can perform on the produced dataset. In turn, this enables to balance the trade off between resource utilization and the total time required to folding the target protein.

As a first order of concern, we verify that the time overheads of EnTK do not depend on scale and that execution concurrency can also be performed without relevant time overheads. Accordingly, we designed Experiment 1 to measure how the total execution time (TTX) of each stage of our pipeline changes across number of resources (compute nodes) and number of tasks concurrently executed in each stage. Further, we measured whether and how the EnTK time management overhead (EOH) varies across scales. Note that EnTK performance has been already characterized and that here we aim at confirming a relevant part of the results already published in in Ref. [24].

Experiment 1 quantifies the relative impact of EnTK on tasks execution and shows how well our runtime system (RADICAL-Pilot) manages execution concurrency. We fix the relation between resources and number of concurrent tasks (i.e., weak scaling), concurrently executing in the first stage 60, 120, 240, 480 and 960 tasks, each using one GPU to execute the OpenMM molecular simulation engine. We do not change the number of tasks on stage 2-4 so to isolate the variations observed by changing the first stage. We execute 1 task in stage 2, 10 tasks in stage 3 and 1 task in stage 4. We execute Experiment 1 on Summit, utilizing between 17 and 280 compute nodes.

Fig.4 shows the TTX and EOH for each stage of the pipeline and across the described scales. In absolute terms, TTX of the first stage (MD TTX) weak scales between 60 and 960 tasks/GPUs. The variation of TTX across scales is minimal: 12498s, 12172s, 12378, 12934s, and . . . for, respectively, 60, 120, 240, 480 and 960 tasks. This indicates that EnTK executed all the tasks concurrently and that the runtime systems added negligible time overheads. EOH is also relatively stable across scales (between 0.79s/. . . and 60/960 tasks) and, in absolute terms, it is negligible when compared to TTX.

EOH is both stable and negligible across the remaining stages of the pipeline. On the contrary, TTX of Stage 2 (Aggregating TTX) increases with scale due to the number of aggregated files but, comparatively, remains irrelevant compared to the TTX of the other stages. The machine learning tasks of Stage 3 (ML TTX) also increases with scale. This likely depends on the amount of data that needs to be processes in order to train the model. The execution time of Stage 4 also varies with scale but seems to peak at 252/70 tasks. After that, the variation in the execution time is just from 5120s to 5145s.

Spending more time to perform the learning and inference tasks is justified only if the quality of the learning and inference processes increases. Increasing the number of MD simulations concurrently executed produces more data and therefore more simulations frames with which to train the ML model. We therefore needed to confirm that the learning and inference processes become more accurate when more data is

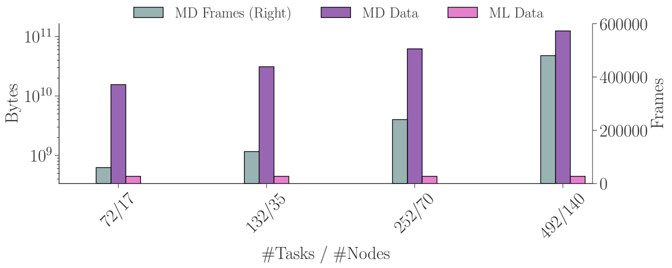

Fig. 5. Total amount of data written by each stage of the workflow and total amount of frames calculated by the tasks of Stage0 as a function of the amount of data produced.

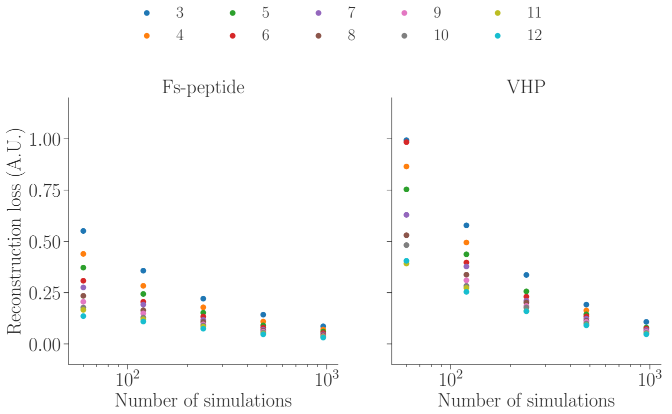

Fig. 6. The concomitant increase in the data gathered from the number of concurrent simulations across GPUs improves the quality of learning. The quality of learning is measured using the reconstruction loss for 1,000 frames from the simulations (as a test data that is withheld from training). We show the results for both Fs-peptide and VHP. The results are measured as a function of the number of latent dimensions learned (varying from 3, . . ., 12).

available.

Fig.5 shows the total amount of data produced by the MD simulations of Stage 1 (purple) and the corresponding amount of frames calculated (gray, plotted again the right y axis). Data and frames grow with the number of tasks while the amount of data produced by the learning process is always the same. To validate if the quality of learning improves with the concurrent increase in training data, we examined the reconstruction loss (explained in more detail in [9]) as a function of both the latent space dimensions and the number of simulation tasks (on GPUs). Note that the reconstruction loss measures how well the latent representation is able to build back the contact matrices after dimensionality reduction using the CVAE. As shown in Fig. 6, as the simulation tasks increase (along with the total number of conformations), the reconstruction loss indeed goes down. It is also remarkable that as the number of dimensions for the latent space representation increases (from 3, . . ., 12), we indeed observe that reconstruction loss decreases – although for higher dimensions, this decrease is less pronounced.

This indicates that increasing the training data available for ML approaches can be particularly useful for our proposed adaptive approach. Although we did not present our complete folding process for VHP here, we observed that even with as little as 0.9μs of aggregate simulations, our adaptive workflow sampled partially folded conformations for VHP. (Note that the folding times for VHP are at the μs timescales).

V. Discussion

Our efforts in this paper were primarily targeted towards understanding the scaling implications of coupling deep learning approaches with MD simulations. We designed two prototypical workflows involving protein folding simulations, capturing typical use cases of how simple ML approaches such as the CVAE can be coupled with such simulations. Indeed for the case of Fs-peptide under implicit solvent conditions, we could demonstrate that the adaptive sampling approaches can sample folded conformations. On the other hand, the set up of our ML-driven MD workflow could not fully fold VHP in explicit solvent conditions within the time allocated on Summit, but was still able to sample partial folding events in its conformational landscape.

Our analysis of the Fs-peptide simulations revealed that only 5 out the 720 (including all simulations from steps 2, 3 and 4 of our iterative workflow in Fig. 1) sampled folded states. A large proportion of these simulations generated the required training data as part of our initialization stage (120 simulations, each with 0.1 μs leading to 12 μs of sampling). After the training (Stage 2 of the workflow), we were able to sample the folded states with less than 6 μs of aggregate sampling. Without any ML, the aggregate sampling required to fold Fs-peptide was 14 μs, which implies that the effective performance [33] gain in sampling using ML based approaches is about 2.33 (14μs to 6μs). Individual simulations in the ML driven workflow were only 0.1 μs in length, as opposed to 0.5 μs in traditional (non-ML) sampling, indicating that by culling unproductive trajectories we can sample the native state of Fs-peptide.

We could have also adopted a previously trained model for our adaptive workflow; however, we explicitly chose not to use such a set up for our experiments. More rigorous tests of the folding times (and kinetics) are required, which we will pursue as part of our future work. This further demonstrates the utility of building ML-driven adaptive MD workflows where some time may be spent initially learning from the running simulations; however, successive iterations of the adaptive workflow can significantly accelerate time-to-solution for expensive simulations.

To overcome computational costs associated with training the deep learning models (i.e., where the limiting factor is the amount of training data available at the start of the workflow), one may use online machine learning tools [34]. However, these tools are limited to analyzing limited data streams from MD datasets and may not result in fully transferable models. Therefore, there is also an explicit need to accelerate training for deep learning approaches [35] as simulations are concurrently running. In addition, the use of one-shot or few-shot learning approaches [20] can be powerful in overcoming the challenges of having to wait for training iterations to complete. We are planning to pursue these approaches as part of our future work.

There are several levels at which DL can be interfaced with MD simulations. At the finest level of granularity, DL models can act as surrogates for simulations. At the highest level of granularity, reinforcement-based DL models can serve to steer the computational campaign towards a pre-determined objective under defined constraints. In between these two different ends of the spectrum, lies the motif of Fig.1 where DL models and methods can be used to guide either individual simulations by determining optimal parameters of exploration, or by intelligently determining regions of phase space to sample, i.e., enhanced sampling. Needless, to say, these three levels are not mutually exclusive and can operate concurrently and collectively to enhanced global computational efficiency, and giving rise to the concept of Learning Everywhere [32], [33] to enhance computational impact. Although this work investigates and focuses on the computational motif in Fig. 1, we will extend capabilities developed here to cover learning integrated with MD simulations at all levels.

References

[1] Stewart A Adcock and J Andrew McCammon. Molecular dynamics: survey of methods for simulating the activity of proteins. *Chemical reviews*, 106(5):1589–1615, 2006.

[2] S. Walter Englander and Leland Mayne. The nature of protein folding pathways. *Proceedings of the National Academy of Sciences*, 111(45):15873–15880, 2014.

[3] Jianyin Shao, Stephen W. Tanner, Nephi Thompson, and Thomas E. Cheatham. Clustering molecular dynamics trajectories: 1. characterizing the performance of different clustering algorithms. *Journal of Chemical Theory and Computation*, 3(6):2312–2334, 11 2007.

[4] Bettina Keller, Xavier Daura, and Wilfred F. van Gunsteren. Comparing geometric and kinetic cluster algorithms for molecular simulation data. *The Journal of Chemical Physics*, 132(7):074110, 2010.

[5] Yan Li and Zigang Dong. Effect of clustering algorithm on establishing markov state model for molecular dynamics simulations. *Journal of Chemical Information and Modeling*, 56(6):1205–1215, 06 2016.

[6] Diwakar Shukla, Carlos X. Hernández, Jeffrey K. Weber, and Vijay S. Pande. Markov state models provide insights into dynamic modulation of protein function. *Accounts of Chemical Research*, 48(2):414–422, 02 2015.

[7] Martin K. Scherer, Benjamin Trendelkamp-Schroer, Fabian Paul, Guillermo Prez-Hernndez, Moritz Hoffmann, Nuria Plattner, Christoph Wehmeyer, Jan-Hendrik Prinz, and Frank No. PyEMMA 2: A Software Package for Estimation, Validation, and Analysis of Markov Models. *Journal of Chemical Theory and Computation*, 11:5525–5542, October 2015.

[8] Arvind Ramanathan, Andrej J. Savol, Christopher J. Langmead, Pratul K. Agarwal, and Chakra S. Chennubhotla. Discovering conformational sub-states relevant to protein function. *PLOS ONE*, 6(1):1–16, 01 2011.

[9] Debsindhu Bhowmik, Shang Gao, Michael T Young, and Arvind Ramanathan. Deep clustering of protein folding simulations. *BMC Bioinformatics*, 19(18):484, 2018.

[10] Helmut Grubmüller, Berthold Heymann, and Paul Tavan. Ligand binding: molecular mechanics calculation of the streptavidin-biotin rupture force. *Science*, 271(5251):997–999, 1996.

[11] Cameron F Abrams and Eric Vanden-Eijnden. Large-scale conformational sampling of proteins using temperature-accelerated molecular dynamics. *Proceedings of the National Academy of Sciences*, 107(11):4961–4966, 2010.

[12] Johannes Kästner. Umbrella sampling. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(6):932–942, 2011.

[13] Zahra Shamsi, Kevin J. Cheng, and Diwakar Shukla. Reinforcement learning based adaptive sampling: Reaping rewards by exploring protein conformational landscapes. *The Journal of Physical Chemistry B*, 122(35):8386–8395, 2018.

[14] Gary A Huber and Sangtae Kim. Weighted-ensemble brownian dynamics simulations for protein association reactions. *Biophysical journal*, 70(1):97–110, 1996.

[15] Peter M Kasson and Shantenu Jha. Adaptive ensemble simulations of biomolecules. 13 September 2018.

[16] Heng Ma, Debsindhu Bhowmik, Hyungro Lee, Matteo Turilli, Michael T Young, Shantenu Jha, and Arvind Ramanathan. Deep generative model driven protein folding simulation. *arXiv preprint arXiv:1908.00496*, 2019.

[17] Raimondas Galvelis and Yuji Sugita. Neural network and nearest neighbor algorithms for enhancing sampling of molecular dynamics. *Journal of Chemical Theory and Computation*, 13(6):2489–2500, 2017.

[18] João Marcelo Lamim Ribeiro, Pablo Bravo, Yihang Wang, and Pratyush Tiwary. Reweighted autoencoded variational bayes for enhanced sampling (rave). *The Journal of Chemical Physics*, 149(7):072301, 2018.

[19] Luigi Bonati, Yue-Yu Zhang, and Michele Parrinello. Neural networks-based variationally enhanced sampling. *Proceedings of the National Academy of Sciences*, 116(36):17641–17647, 2019.

[20] Frank Noé, Simon Olsson, Jonas Köhler, and Hao Wu. Boltzmann generators: Sampling equilibrium states of many-body systems with deep learning. *Science*, 365(6457):eaaw1147, 2019.

[21] R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, , T. F.G. Green, C. Qin, A. Zidek, A. Nelson, A. Bridgland, H. Penedones, H. Petersen, K. Simonyan, S. Crossan, D.T. Jones, D. Silver, K. Kavukcuoglu, H. Hassabis, and A.W. Senior. De novo structure prediction with deep learning based scoring. In *Thirteenth Critical Assessment of Techniques for Protein Structure Prediction*, 2018.

[22] Matteo Turilli, Andre Merzky, Thomas Naughton, Wael Elwasif, and Shantenu Jha. Characterizing the performance of executing many-tasks on summit. *arXiv preprint arXiv:1909.03057*, 2019.

[23] Vivek Balasubramanian, Antons Trekalis, Ole Weidner, and Shantenu Jha. Ensemble Toolkit: Scalable and Flexible Execution of Ensembles of Tasks. In *Proceedings of the 45th International Conference on Parallel Processing (ICPP)*, 2016. http://arxiv.org/abs/1602.00678.

[24] Vivek Balasubramanian, Matteo Turilli, Weiming Hu, Matthieu Lefebvre, Wenjie Lei, Ryan Modrak, Guido Cervone, Jeroen Tromp, and Shantenu Jha. Harnessing the power of many: Extensible toolkit for scalable ensemble applications. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 536–545. IEEE, 2018.

[25] Andre Merzky, Matteo Turilli, Manuel Maldonado, Mark Santcroos, and Shantenu Jha. Using pilot systems to execute many task workloads on supercomputers. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 61–82. Springer, 2018.

[26] Andre Merzky, Ole Weidner, and Shantenu Jha. SAGA: A standardized access layer to heterogeneous distributed computing infrastructure. *Software-X*, 2015. DOI: 10.1016/j.softx.2015.03.001.

[27] Tom Goodale, Shantenu Jha, Hartmut Kaiser, Thilo Kielmann, Pascal Kleijer, Andre Merzky, John Shalf, and Christopher Smith. A Simple API for Grid Applications (SAGA). OGF Recommendation, GFD.90, Open Grid Forum, 2007.

[28] Matteo Turilli, Mark Santcroos, and Shantenu Jha. A comprehensive perspective on pilot-job systems. *ACM Comput. Surv.*, 51(2):43:1–43:32, April 2018.

[29] Matteo Turilli, Vivek Balasubramanian, Andre Merzky, Ioannis Paraskevakos, and Shantenu Jha. Middleware building blocks for workflow systems. *Computing in Science & Engineering (CiSE) special issue on Incorporating Scientific Workflows in Computing Research Processes*, 2019.

[30] Vivek Balasubramanian, Shantenu Jha, André Merzky, and Matteo Turilli. Radical-cybertools: Middleware building blocks for scalable science. *CoRR*, abs/1904.03085, 2019.

[31] Shantenu Jha, Scott Lathrop, Jarek Nabrzyski, and Lavanya Ramakrishnan. Incorporating scientific workflows in computing research processes. *Computing in Science and Engineering*, 21(4):4–6, 2019.

[32] Geoffrey Fox and Shantenu Jha. Understanding ml driven hpc: Applications and infrastructure. *arXiv preprint arXiv:1909.02363*, 2019.

[33] Geoffrey C. Fox, James A. Glazier, J. C. S. Kadupitiya, Vikram Jadhao, Minje Kim, Judy Qiu, James P. Sluka, Endre Somogy, Madhav Marathe, Abhijin Adiga, Jiangzhuo Chen, Oliver Beckstein, and Shantenu Jha. Learning everywhere: Pervasive machine learning for effective high-performance computation. In *IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 422–429, 2019. https://arxiv.org/abs/1902.10810.

[34] Arvind Ramanathan, Ji Oh Yoo, and Christopher J. Langmead. On-the-fly identification of conformational substates from molecular dynamics simulations. *Journal of Chemical Theory and Computation*, 7(3):778–789, 03 2011.

[35] Srikanth B. Yoginath, Maksudul Alam, Arvind Ramanathan, Debsindhu Bhowmik, Nouamane Laanait, and Kalyan S. Perumalla. Towards native execution of deep learning on a leadership-class HPC system. In *IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019, Rio de Janeiro, Brazil, May 20-24, 2019*, pages 941–950, 2019.