Employing Artificial Intelligence to Steer Exascale Workflows with Colmena

Logan Ward,[1] J. Gregory Pauloski,[2] Valerie Hayot-Sasson,[2] Yadu Babuji,[2] Alexander Brace,[2] Ryan Chard,[1] Kyle Chard,[2] Rajeev Thakur[1] and Ian Foster[1]

Abstract

Computational workflows are a common class of application on supercomputers, yet the loosely coupled and heterogeneous nature of workflows often fails to take full advantage of their capabilities. We created Colmena to leverage the massive parallelism of a supercomputer by using Artificial Intelligence (AI) to learn from and adapt a workflow as it executes. Colmena allows scientists to define how their application should respond to events (e.g., task completion) as a series of cooperative agents. In this paper, we describe the design of Colmena, the challenges we overcame while deploying applications on exascale systems, and the science workflows we have enhanced through interweaving AI. The scaling challenges we discuss include developing steering strategies that maximize node utilization, introducing data fabrics that reduce communication overhead of data-intensive tasks, and implementing workflow tasks that cache costly operations between invocations. These innovations coupled with a variety of application patterns accessible through our agent-based steering model have enabled science advances in chemistry, biophysics, and materials science using different types of AI. Our vision is that Colmena will spur creative solutions that harness AI across many domains of scientific computing.

Keywords

workflows, artificial intelligence, computational steering

Introduction

Decades of steadily improving computer hardware have made computers often faster at answering questions than humans are at posing them. As such, artificial intelligence (AI) algorithms are playing an increasing role in science as both programmer and software. Supervised learning algorithms improving approximate models for costly simulations without human direction, language models generating code that answers questions posed by humans as general questions, and many other marvels are commonplace. Under this context, the future of computational workloads may be filled with self-directed software.

Workflows, applications that orchestrate execution of many diverse tasks, have been a major source of innovation in AI for high-performance computing (HPC) (Ferreira da Silva et al. 2024). The recurring nature of tasks provides the consistent, easily defined training sets that make AI integration simpler. Key examples of AI in workflows include experimental design techniques that infer best inputs given the history of results (Jacobsen et al. 2018), unsupervised learning techniques to draw inferences from output data on-the-fly (Lee et al. 2019), or generative techniques that invent what search spaces to explore (Gómez-Bombarelli et al. 2018). The growing range and increasing intelligence of AI models suggests these examples are an early example of a space filled with opportunity.

Optimal performance of the AI within an application relies on accounting for nuances in how it is used. As an illustrative example, the AI tasks in an experimental design workflow only need to be performed when "sufficient" data are required, and the notion of "sufficient" depends on many aspects of the application. AI models that are fast compared to the tasks they advise could be run as each task is completed, whereas relatively expensive AI models should be delayed and deployed on dedicated resources. Expensive AI models may also benefit from a streaming policy where simulations are started based on intermediate results of the AI tasks, rather than waiting for all to complete. Such ideas for harmonically composing simulation and AI tasks are still growing.

We developed Colmena to allow scientists to create inventive solutions for integrating AI into workflow applications on supercomputing systems (Ward et al. 2021). Colmena is a Python library that adds a layer to conventional workflow systems that simplifies expressing the dynamic ways AI can be used. In this paper, we start by describing the previous work that inspired and enabled Colmena, then introduce its implementation before discussing a few HPC case studies.

Related Work

Our work sits at the intersection of AI and scientific workflows, providing a unique approach to combining them.

[1]Argonne National Laboratory, IL, USA
[2]University of Chicago, IL, USA

Corresponding author:
Logan Ward, Data Science and Learning Division,
Argonne National Laboratory, Lemont, IL, USA
Email: lward@anl.gov

AI Approaches for Science Workflows

There are numerous modalities for how AI can be used in simulation, and they can be understood by the relationship between AI and the simulation software it enhances (Fox et al. 2019). The purpose of the AI could be to post-process the result of simulations and synthesize the outputs, as in the learning of collective variables in DeepDriveMD (Lee et al. 2019; Brace et al. 2022). AI could also be used to create a fast surrogate that can be used to prejudge potential simulations from within a pre-defined search space (St. John et al. 2019) or to generate potential simulations that resemble previous successes (Huerta et al. 2023). Each different relationship implies a different order of operations in how the simulation and AI components are combined.

The strength of coupling between the tools and the relative degree of computational costs between AI and simulation impose further constraints on how applications are implemented. Even within the archetype of "AI used to create surrogates," distinct variations are possible. Applications may use the AI and simulation code concurrently, which requires low-latency inference from the ML models and therefore a need for dedicated resources for both computations (Caccin et al. 2015). In contrast, one that uses AI and simulation sequentially can use the same resources for both (Zamora et al. 2021). Similar variations in degrees of coupling exist across other AI and simulation archetypes, yielding high diversity in the ways AI and simulation should be combined together.

Workflow Engines

Workflow engines and other related approaches to the task-parallel, often data-driven, execution of tasks on HPC systems has been a subject of research since at least the 1980s, when systems like Linda (Carriero and Gelernter 1988) and Strand (Foster and Taylor 1989) were used for such purposes, while Condor (Thain et al. 2005) and Condor-G (Frey et al. 2002) enabled dispatch of many tasks within one or multiple resource pools, respectively. Parallel scripting approaches enable efficient execution of tasks coupled by file system operations (Zhao et al. 2005; Wilde et al. 2009).

AI-enhanced applications provide challenges to the workflow engines that manage their computations on HPC. The AI-based workflows that we consider here continuously update the tasks to be executed during the execution of the workflow. This is in contrast to the traditional static approach employed by many workflow engines that require the entire workflow graph to be defined a priori (Liu et al. 2015; Herath and Plale 2010). Further, this dynamicity requires efficient workflow processing as the latency of tasks being created to running on a worker becomes as important as task throughput. Tasks in AI-based workflows are particularly heterogeneous, with AI training tasks spanning many nodes while inference tasks require fractions of individual GPUs (Dhakal et al. 2023). Such heterogeneity presents many challenges in efficiently scheduling tasks across available nodes (Phung et al. 2021). It also creates new opportunities, for example, to deploy workflows that span multiple types of compute resources. In short, the addition of AI into scientific workflows will push the state-of-the-art on workflow engines.

Integrating AI and Workflows

The variety of ways AI can be used in simulation has led to a bloom of approaches to implement them in software. This emerging class of tools built to support workflows that include AI, which includes Colmena, share a common set of features:

- *Templating for common design patterns*, such as how AI applications are broken down into generator, simulator, and allocator by libEnsemble (Hudson et al. 2022) or the problem definition templates of DeepHyper (Balaprakash et al. 2018).
- *Dedicated services or tasks for running AI tasks*, such as the persistent optimization server in CANDLE Supervisor (Wozniak et al. 2018) or the "OptTask" in RocketSled (Dunn et al. 2019).
- *Integration with data fabrics*, such as via the use of SmartRedis to transmit data to AI tasks in SmartSim (Partee et al. 2022) or the two choices of data fabrics in eFlows4HPC (Ejarque et al. 2022).
- *Connections to machine learning frameworks*, as illustrated by ties to PyTorch and Tensorflow within the hyperparameter tuning and training extensions for Ray (Moritz et al. 2018).
- *Emphasis on dynamic allocation* to accommodate changes in tasks types as AI models improve, visible in the integration of dynamic schedulers in DeepDriveMD (Lee et al. 2019).

While similar along these dimensions, the tools vary significantly in how they define and deploy workflows. For example, there is a divide between stateful processes with decentralized communication (e.g., SmartSim, Ray, Decaf (Yildiz et al. 2021)) and others with a central controlling process (e.g., Supervisor, Colmena). Learning how best to grow these ranges of capabilities alongside the diversifying landscape of AI workflows remains an open question.

Design

The purpose of Colmena is to write policies that schedule computation and data movement as Python functions. We chose Python functions to allow policies of arbitrary complexity to be expressed in a well-known language. Details for distributing computations are delegated to other libraries. Colmena applications are composed of two components: a Thinker and Task Server (see application design, Figure 1). We start by describing how policies are written as a Thinker and then detail how Colmena Task Servers use third-party tools to execute applications at scale.

Programming Model

A Thinker is a Python object whose methods define the policy of a computational campaign. Methods marked with special decorators run as threads after a user invokes the Thinker. The threads, which we refer to as agents, submit computations to and receive results from a Task Server (described in detail in Task Execution) over a shared queue.

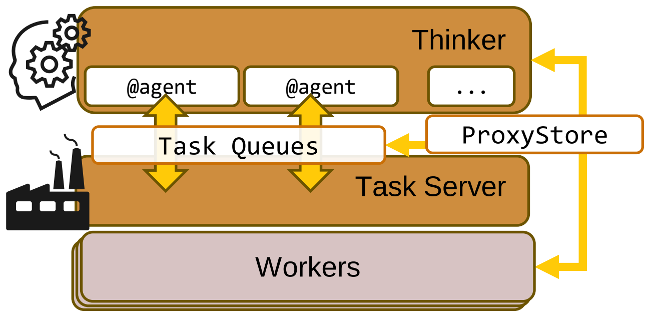

Figure 1. A Colmena application is composed of a Thinker and Task Server connected by a task queue. Thinkers define the policy for submitting computations using a series of agents that interact with each other and the Task Servers. Task Servers delegate computations to workers running on compute nodes. Applications that manage large datasets or run at large scales can use ProxyStore to pass references to inputs and outputs via the workflow engine and object data via a side channel.

Interactions between the agents and the results of tasks control how a computational campaign evolves.

Listing 1 illustrates an example Thinker that implements a Markov Chain Monte Carlo sampling algorithm with two agents. The *startup* agent submits a population of tasks as soon as the run method is called and then exits, while the *step* agent receives completed tasks and, for each, submits a new task so as to maintain a constant amount of work on the supercomputer.

Using Python functions to express the steering logic ensures room to design sophisticated strategies. One could, for example, add a third agent that manages training a surrogate model and augment *step* to use the surrogate model when reasonable, or introduce logic to restart sampling trajectories that become trapped in already-observed states, as in DeepDriveMD (Lee et al. 2019; Brace et al. 2022). Colmena leaves avenues for optimization open.

Agent Types Colmena supports four types of agents that each fulfill common tasks in a workflow steering policy:

1. `@agent` starts at the beginning of an application and is expected to run until the end of the application unless marked with a "startup" option.
2. `@result_processor` runs when a task of a certain type completes and is provided the result object (see Defining Tasks) as an input.
3. `@event_processor` is invoked when an associated Python Event object is set.
4. `@task_submitter` executes when a certain number of resources are available. Resources are tracked using a set of semaphores that can be accessed by all agents.

Threading We use Python's standard threading library to run agents in parallel and to coordinate between the agents. For example, the resource tracking used by some Thinkers to balance the number of tasks of each type employs Python's built-in Semaphore objects. Using standard libraries makes writing a Thinker as close to standard Python as possible.

The Global Interpreter Lock (GIL) of Python has yet to become a major limitation of Colmena applications. The steering agents are intended to be lightweight, completing in just a few milliseconds for our case study applications

```python
from colmena.thinker import BaseThinker
from numpy.random import random, sample
import numpy as np


class Thinker(BaseThinker):

    def __init__(
        self,
        queues,
        dimensionality: int = 8,
        num_samples: int = 256,
        walkers: int = 8,
    ):
        super().__init__(queues)
        self.n = num_samples
        self.d = dimensionality
        self.samples = []
        self.x = \
          sample((walkers, self.d)) * 2 - 1
        self.log_p = \
          np.zeros((walkers,)) + np.inf

    @agent(startup=True)
    def startup(self):
        for i, x in enumerate(self.x):
            self.queues.send_inputs(
                x,
                method='compute_logp',
                task_info={'w': i},
            )

    @result_processor()
    def step(self, result: Result):
        # Perform MC step
        w = result.task_info['w']
        new_lp = result.value
        old_lp = self.logp[w]
        accept = np.exp(new_lp - old_lp) \
            < random()
        if accept:
            self.logp[w] = new_logp
            self.x[w, :] = result.args[0]

        # Submit a new sample, if not done
        if not self.done.is_set():
            self.queues.send_inputs(
                self.x[w] + \
                  random((self.d,)) * 2 - 1,
                method='compute_logp',
                task_info={'w': w},
            )

        # Store, then stop if done
        self.samples.append(self.x[w])
        if len(self.samples) > self.n:
            self.done.set()

thinker = Thinker(queues)
thinker.run()
```

Listing 1. A Colmena Thinker that implements a parallel Metropolis-Hasting algorithm. The Thinker stores the positions of each walker, the current probability for each position, and the output samples. The *startup* agent submits an initial set of computations then exits. The *step* agent runs when a computation finishes, updates the state of the associated walker, then submits a computation for the next point.

(Ward et al. 2021, 2023). Even if no agent may operate concurrently (i.e., if no parts of the operation release the GIL), a millisecond processing time per task places a general limit of thousands of actions per second—large enough for many applications. We have considered adding the ability for some agents to run as separate Processes, which are free from GIL considerations but such a design would make it harder to coordinate with other agents.

Defining Tasks

The computations requested by Colmena Thinkers, *Tasks*, are defined as Python functions. As in other workflows, function definitions must be serializable (true for all functions defined in modules) and take inputs that can be serialized. There is a great variety available within these bounds. The tasks may be pure Python and run on a subset of a compute node, or make calls to external applications that span many compute nodes. So long as they are defined via a Python interface, Colmena can run them.

The Thinker application requests a task using the name of the function and a series of positional or keyword arguments, as if calling the function locally. The Thinker then prepares a Result object which captures the input information as well as any other information needed to define the task, such as resource requirements (e.g., a number of processors) or task metadata that would be useful in processing results later (e.g., an identifier connecting similar tasks). Task Servers will populate the Result object with the results of the computation as well as communication overheads and execution times so that users can adjust subsequent tasks accordingly or, at least, analyze performance afterward.

Task Queues

The Task Queue communicates task requests and completed results between Thinker and Task Server. A single application may use separate Task Queues for different classes of tasks so that groups of agents can operate independently. Applications can also use different Queue implementations. Redis, for example, is well suited for large task rates or data sizes, but the complexity of running a Redis Server may not be justified in some cases, compared to Python's built-in Pipes. All queues, regardless of type, use the same interface so that it is simple to exchange components to port an application between different computing systems or scales.

Task Execution

The Task Server stewards the execution of tasks requested by the Thinker. The Task Server interface provides an abstraction over arbitrary workflow engines and is responsible for translating task requests from a Thinker, dispatching tasks to the workflow engine, and returning completed requests to the queue. This abstraction ensures that Colmena applications are portable; the specific Task Server implementation can be exchanged allowing the same Thinker to run on different HPC systems.

Colmena provides Task Server implementations for Parsl (Babuji et al. 2019) and Globus Compute; other Task Server implementations can also be developed. Parsl and Globus Compute can run arbitrary Python functions on arbitrary compute resources, from laptops to the largest supercomputers. Parsl provides multiple types of executors suitable for different use cases and has shown to scale to workloads up to thousands of tasks per second. Globus Compute is a cloud-managed, federated function-as-a-service platform. The cloud-managed infrastructure enhances reliability and makes it simpler to deploy applications across multiple sites, in contrast to Parsl, which requires additional network configuration to use multiple compute resources concurrently. Succinctly, Globus Compute trades task throughput, latency, and reliability for easy access to remote compute resources, although we note the performance tradeoffs are small for most workloads (Ward et al. 2023).

Data Fabric

Tasks in AI-centric applications often consume or produce copious amounts of data, which can lead to nontrivial communication overheads. For example, all task data flows through the Task Server process in Colmena, so data-intensive tasks can result in heavy I/O burdens that slow down the process. This challenge is not unique to Colmena as most workflow engines have a central coordinator, such as the Globus Compute cloud service, through which all task data must be transferred. We provided tools in Colmena to circumvent these I/O bottlenecks.

Colmena integrates with ProxyStore (Pauloski et al. 2023, 2024) to reduce communication overheads by moving data through specialized channels rather than through the Task Server and workflow engine. ProxyStore replaces Python objects with *proxies* that reference the location of the actual data and then resolve to the original object when used. The proxy is small, making it suitable to transmit alongside the control messages of the workflow engine while the object data are propagated to workers using better-suited communication protocols (e.g., Redis, Globus, Remote Direct Memory Access). In essence, ProxyStore translates task data from being passed-by-value to passed-by-reference, avoiding unnecessary copies of data across processes or expensive serialization. These proxies provide other benefits: proxies can be asynchronously resolved at the start of task execution to overlap compute and I/O, and I/O costs are not incurred for large objects when tasks exit early or fail unexpectedly.

Colmena can make use of ProxyStore in two different ways: by configuring the Task Queues to automatically proxy large task objects and by proxying objects manually in the Thinker before task submission. The two methods can be employed at the same time; in neither case do the Python functions comprising the Colmena tasks need to be modified.

Configuring the Task Queues to automatically proxy objects is the simplest way to obtain potential performance benefits. In this approach, the user configures ProxyStore with parameters such as the communication protocols to be used, and passes that configuration to the queues. Task objects (positional and keyword arguments) are then replaced automatically with proxies when a new task is created, and the result of a task is proxied automatically after execution. However, this automated approach provides limited routes for optimizing transfer performance. Thus it can also be beneficial to proxy objects manually, for example when an object is used by many tasks. In this case, the user may create a proxy within the Thinker and pass that proxy as a task argument.

Similar to how the Task Server abstraction makes it simple to redeploy a Colmena application on a different workflow engine or HPC system, ProxyStore decouples the configuration of communication protocols used to transmit task data from the application. This reduces friction when migrating applications across systems with different networking or storage stacks. No application code needs to be changed—only the ProxyStore configuration.

Scaling on Supercomputers

Workflow engines such as Parsl permit most applications to scale to dozens of nodes without special effort. The applications described in the section on Other Successes taught the Colmena development team many strategies for accessing scales in the hundreds or thousands of nodes:

1. *Passing data by reference* is critical for tasks that involve data larger than $O(100)$ kB or workflows that span more than one system (Ward et al. 2023). Passing large data via the workflow system leads to communicating tasks to compute nodes becoming a bottleneck. Object proxies are a powerful solution to such problems because task code need not be changed to resolve references, and caching accelerates tasks that reuse data, such as inference tasks that use the same model over many input batches.
2. *Avoiding unnecessary reinitialization* across functions can accelerate workflows. It is common for multiple tasks executed on the same worker to re-initialize the same expensive objects because workflow engines are designed to work with pure functions. In other words, the workers are not stateful actors. We circumvent this, for example, by keeping lookup tables or machine learning models used by tasks in RAM when not in use, rather than loading them from disk each time (Dharuman et al. 2023).
3. *Acting on task completion rather than result reception* is possible in cases when a task finishing could inform the creation of a new task without the need to receive or process results yet. Employing ProxyStore to separate control messages and data flow means that result notifications can be received two orders of magnitude sooner, which can be exploited to hide the latency of data transfer (Ward et al. 2023; Harb et al. 2023). That is, we can act on a task finishing and defer processing the results until result data are available.

Case Study: Molecular Design

We used the design of molecules for redox-flow batteries as the prototype application for Colmena (Ward et al. 2021). The application runs tasks that compute the performance of a molecule (i.e., solvation energy, redox potential), train a model that predicts performance quickly, or infer the performance of new molecules. As in other examples of AI-driven design (Doan et al. 2020; Montoya et al. 2020; Badra et al. 2022; Curtarolo et al. 2003; Zhang et al. 2020), our Colmena application achieves orders-of-magnitude improvements over unguided searches. The advantage of Colmena is that we could access another 20% increase in the number of high-performing molecules found by co-scheduling simulation and AI tasks (see Figure 2). Achieving this increase in scientific performance while also maintaining effective use of the HPC resources required many innovations, addressing challenges at different parts of the application.

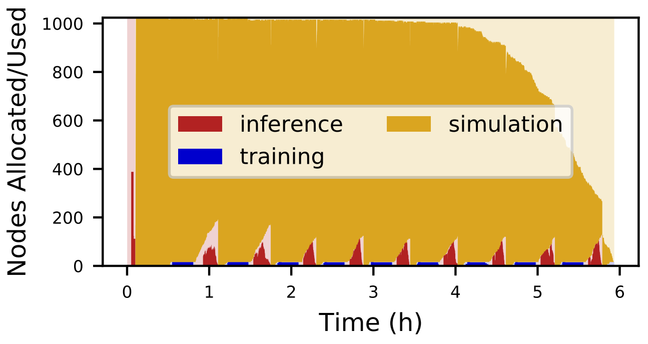

Figure 2. Allocation of HPC nodes between different tasks over time for a Colmena-based molecular design application. Nodes may either run quantum chemistry simulations (yellow), train a machine learning model (blue), or use the model to infer the properties of a molecule (red). The application first runs inference on all nodes and then runs simulation tasks until sufficient data is available to begin re-training machine learning and re-running inference on a subset of nodes. Light shades indicate periods where either no computation was running or the running calculation did not complete before the end of the allocation. Figure from Ward et al. (2021).

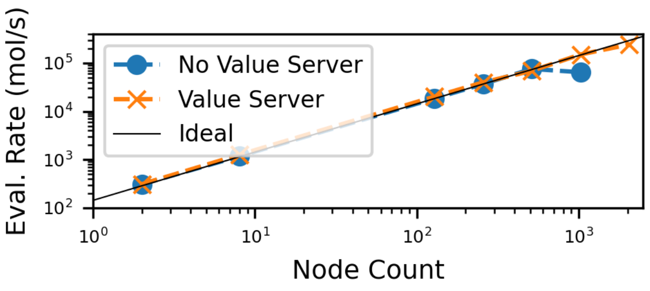

Figure 3. Weak scaling of inference rate as a function of node count for a molecular design application that uses message passing neural networks. Experiments were performed on the Theta Supercomputer at Argonne National Laboratory. Figure from Ward et al. (2021).

Reducing Communication Overheads

The initial step in our molecular design application is to identify a set of target molecules by running inference over all molecules. Inference tasks transmit large amounts of data (copies of the models, molecules, inference results) between many nodes, which presents a clear challenge to scaling. Large scales require faster transfer rates to keep all nodes populated with work and, as visible in Figure 3, data transfer became rate-limiting at only 512 nodes.

We identified the connection between the Task Server and worker nodes as the source of the problem. Result data would remain on a compute node for as long as 10 seconds, which signals that the channels exchanging control messages between the workflow engine and workers have become saturated. We alleviated the communication backlog by removing as much data from the control messages as possible and passing it instead through a “Value Server” (now available as ProxyStore (Pauloski et al. 2023)) that routes data directly between the Thinker and compute nodes.

We integrated ProxyStore into Colmena as part of the Task Queues. The new queue intercepted inputs larger than a chosen size (10 MB in our case), stored their serialized representation in a Redis instance running on the same node as the Task Server, and replaced them with a reference. A similar swap occurs on the compute node for result objects larger than a certain size. Neither modifications to Colmena require changes to the source codes of the tasks and, as shown in Figure 3, improved the scaling limit to above 2000 nodes.

Using Specialized Hardware for AI Tasks

Our AI-based optimization algorithm works best when the time between acquiring new data and updating recommendations for new simulations is minimized. Updating recommendations requires retraining machine learning models and then rerunning inference, a task that requires about 45 minutes on the CPU-only nodes on the Theta supercomputer at Argonne National Laboratory (see Figure 2). We shortened this time by offloading machine learning computations to a GPU cluster, which required solving an amplified set of problems in data transfer (Ward et al. 2023).

We augmented the Colmena and Value Server framework used in Ward et al. (2021) to provide multi-resource compute and secure data transfer between resources via Globus. An earlier version of Colmena required users to maintain SSH tunnels on at least three ports (two for the workflow engine, and one for the Value Store), which increases maintenance burden and may be disallowed on some systems. Our next implementation used Globus Compute—previously known as FuncX (Chard et al. 2020)—to route task requests through a cloud service and Globus Transfer to move task data between systems. Communicating tasks through Globus Compute required 100 ms and performing data transfer required at least 1 s: higher latencies than with our Parsl-based implementation but still small enough to mitigate.

We hid the data transfer latency by modifying the Thinker to use bulk transfers of task data in advance of tasks being executed. Such optimized transfers are performed by manually creating proxies for task data rather than relying on the automated mechanisms introduced in Ward et al. (2021). Our molecular design app uses bulk transfer at several points: the molecules used in inference tasks once at the beginning of a run, the data used for training tasks each time model training starts, and the models used for inference as soon as model training completes. Enacting the transfers manually reduces the number of times a transfer must be started (at a cost of 1 s each) and manually creating the proxy allows reusing them between tasks. Reuse is visible in data transfer, accounting for less than 1% of execution time for many inference tasks during a run.

The new data transfer coupled with the ability to hide latencies with tailored steering policies makes the convenience of cloud services available without

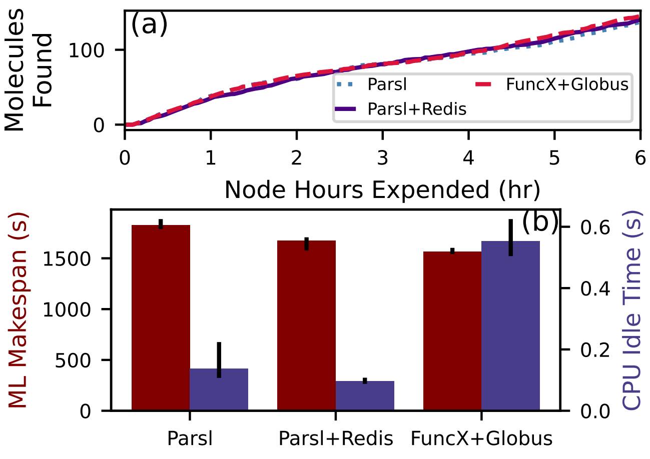

Figure 4. (a) Scientific output of our molecular design application over time and (b) key performance timings (time to complete machine learning tasks, average time between tasks for CPU workers) of a multi-site implementation of our molecular design application with different Colmena backends. Our implementations using the Parsl workflow engine and Parsl with Redis to transmit task data both required maintaining SSH tunnels between sites. The implementation with FuncX and Globus does not require direct network connections between sites, yet has similar scientific output and comparable performance timings.

performance penalties. Figure 4 shows that scientific output is unharmed by using a more resilient, Globus-Compute-based implementation, and the makespan of the machine learning tasks is even better. We attributed the increased performance in machine learning to the ahead-of-time transfer and noted that the increased latency in CPU idle time does not reduce overall utilization below 99%. In short, we made Colmena a suitable tool for workflows that can benefit from differing types of hardware for each task.

Better Performance through Integrating More AI

The large number of trailing tasks at the end of Figure 2 signals an opportunity: we can gain performance by breaking tasks into smaller parts. Beyond the gains of just ensuring less information is lost at the end of an allocation, introducing smaller tasks provides more opportunities for making smarter decisions with AI. More decision points create both more tasks and sensitivity to latency in waiting for decisions, leading to a stronger need for highly parallel computing systems. We consequently are exploring finer granularity of AI and simulation tasks for the exascale version of our molecular design application.

We build multiple steps into our simulation workload by introducing multiple levels of fidelity. Rather than compute the performance of a molecule at the target level of accuracy, as in Ward et al. (2021), we now perform parts of the calculation incrementally (e.g., one property before another) and use multiple levels of accuracy for our simulation codes (i.e., smaller basis sets, cheaper DFT functionals). As demonstrated by Woo et al. (2023) and Reyes et al. (2022), these additional steps in accuracy reduce the cost of an optimization algorithm because it is possible to stop evaluating low-performing candidates before incurring the full computational cost.

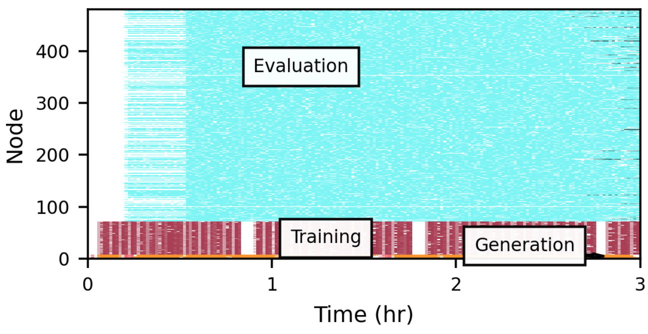

Figure 5. Utilization of each of 480 nodes (1920 GPUs) of ALCF's Polaris supercomputer over time for a Colmena application that simultaneously **trains** a reinforcement learning model that generates proteins, **generates** new proteins with that model, and **evaluates** the quality of the proteins. Periods of higher utilization are represented as deeper shades and color indicates the type of task being run.

Ongoing Challenges

A few sources of performance degradation remain elusive.

The under-utilization during the first minutes of Figure 2 and during each subsequent batch of inference tasks is a result of the delay in loading Python libraries. Speeding the library load rate can be accomplished by reducing the number of reads from the global filesystem (Kamatar et al. 2023). We have yet to be able to make such solutions accessible to Colmena applications or the workflow engines that underlie them.

Individual molecule simulation tasks may involve launching an MPI executable multiple times, which can lead to significant overheads on large systems. We have studied this problem using Colmena as a use case (Alsaadi et al. 2022) and intend to continue participating in the workflow community to adopt the latest advancements.

Ensuring each node running computations is used to its full extent is challenging because of the variety of tasks in an AI workflow and potential variance in resource needs within tasks of a single type. The molecular design application described here has served as a test case for exploring systems that identify automatically how to partition individual nodes for multiple tasks (Phung et al. 2021) and for evaluating the effect of partitioning individual compute units within a node for tasks (Dhakal et al. 2023). We continue to investigate methods for node partitioning.

Other Successes

We have implemented various applications using Colmena since our first prototypes in mid-2021 (Ward et al. 2023; Dharuman et al. 2023; Harb et al. 2023), each highlighting different challenges and opportunities for braiding AI into workflows. We discuss two of these applications below.

Protein Generation

The large machine learning models central to the success of the protein design work of Dharuman et al. (2023) were the primary source of scaling challenges. The core of this application is a genome-scale language model (GenSLM) which produces genetic sequences that are then filtered to find the best-performing sequences (Zvyagin et al. 2023). Then, another large language model (Lin et al. 2023) is used to fold the translated protein sequence, and a series of simulation steps are applied to the folded protein. With Colmena, we could deploy all of these disparate task types together in a manner that balanced maximizing HPC utilization with scientific performance.

The central tradeoff of the protein design workflow is that generating or processing sequences without interruption maximizes system utilization, but more frequent reporting improves algorithm performance through better information flow. As shown in Figure 5, the largest periods of under-utilization in our application after the cold-start phase are the periods where the reinforcement learning training is being stopped and then restarted to update the version of the model used for generation. Balancing this tradeoff required designing a Colmena application that lowered the cost of reporting through the following strategies:

- *Performing CPU-bound tasks asynchronously* from the GPU-intensive machine learning and simulation tasks. The Thinker application for Colmena was designed to process task results only after launching new tasks, ensuring that GPU tasks were dispatched with minimal delay.
- *Caching large models in CPU memory* was necessary to allow tasks that both use large amounts of GPU models (e.g., protein folding, diversity scoring) to share the compute node. Keeping the model saved in memory increased the throughput of folding tasks by 30% and prevented the nodes used for folding from being idled while downstream tasks were completed.
- *Minimizing access to global filesystems* by having any task write intermediate data to local temporary storage (e.g., RAM disk) during computation, then passing completed results in-memory via ProxyStore rather than relying on the global filesystem to transmit results. Even though the data is serialized twice (once to local, once to ProxyStore), limiting the frequency of writes to the global filesystem is worthwhile.

A dynamic workflow system, such as Colmena, simplified expressing each of these strategies.

Our work with this application has allowed us to identify avenues for future improvements in Colmena. Adding the ability for Colmena to report intermediate results would provide the largest improvement by eschewing the need to restart tasks. Integrating Colmena with in-situ workflow tools such as Decaf (Yildiz et al. 2021) or with streaming systems such as Flink (Carbone et al. 2015) could allow for AI tasks to act as standalone services, while simulation tasks are served by a workflow system. Expressing AI applications that blend workflows centered on atomic tasks and persistent services is a research area ripe for exploration.

Steering Molecular Dynamics

Molecular dynamics (MD) simulation of complex biomolecular systems is a prominent HPC application (Casalino et al. 2021; Dommer et al. 2023; Trifan et al. 2022; Phillips et al. 2002) that acts as a computational microscope (Dror et al. 2012) revealing biophysical details difficult to observe via experiment. Due to high free energy barriers, many important

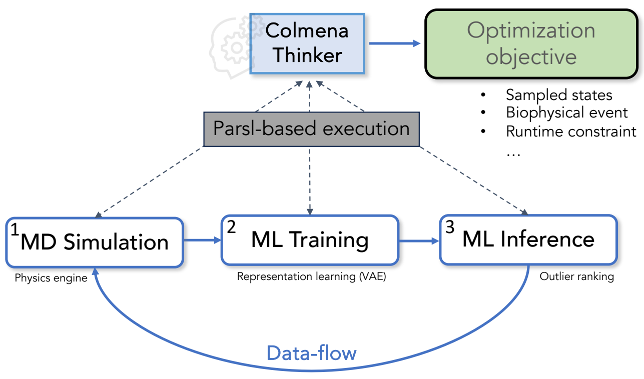

Figure 6. DeepDriveMD accelerates the sampling of rare biophysical events within molecular dynamics (MD) simulations. A Colmena Thinker orchestrates an ensemble of MD simulation tasks while asynchronously training a machine learning (ML) model such as a variational autoencoder (VAE). ML inference and outlier detection are used to guide the simulation state sampling towards a customizable optimization objective.

phenomena are difficult or impossible to sample using conventional MD, even with powerful supercomputers (Hospital et al. 2015). To approach this problem, Lee et al. (2019) and Brace et al. (2022) developed the DeepDriveMD framework, illustrated in Figure 6, for coupling ML/AI methods to MD simulations to track the simulated state space and guide simulations to sample more biophysically interesting events, constituting rare events.

The design pattern underlying DeepDriveMD involves steering an ensemble of many simulations using a trained ML model, such as a variational autoencoder (Bhowmik et al. 2018), for inference. To keep the model up to date with the new data coming from the simulations, it must be periodically retrained. However, updating the model necessitates a speed versus accuracy tradeoff. Making immediate decisions on which simulations to stop or continue requires the use of a stale model (i.e., one that is not completely up-to-date). On the other hand, the delay induced by training may enable more accurate decisions that could better explore the simulation state space and ultimately lead to faster convergence for rare-event sampling.

DeepDriveMD, as a representative example of the Colmena steering paradigm, illustrates several broadly applicable strategies:

- *Asynchronous simulation and ML training* decreases the lead time for training ML models over a synchronized execution pattern, as the ML training is not blocked by simulation stragglers. In addition, hardware accelerators can be employed to further reduce the training time (Brace et al. 2021).
- *Streaming simulation data* for training and inference minimizes the file system I/O of the workflow, which is important for attaining high performance on leadership-scale facilities beyond 1,000 nodes.
- *On-demand ML inference* allows the Thinker application to make decisions to stop or continue simulations without having to wait for the latest training task to finish, which leads to higher resource utilization and more MD sampling throughout a campaign.

While DeepDriveMD has shown up to 100–1000$\times$ speedups for sampling protein folding pathways of certain biomolecular systems (Brace et al. 2022), we note that success is largely dependent on the proper alignment of the ML/AI method and the simulation data being generated. Hyperparameter tuning can play a large role in determining the success of sampling rare events, and domain-specific biophysical calculations are still needed to guide AI-driven sampling properly. Incorporating hyperparameter tuning frameworks such as DeepHyper (Balaprakash et al. 2018) may increase the robustness of AI-steered simulation workflows. Furthermore, uncertainty quantification through model ensembling (Egele et al. 2022), reinforcement learning adaptions (Shamsi et al. 2018), and incorporating statistically rigorous weighted ensemble approaches (Russo et al. 2022) represent promising future directions for such workflows.

Next Steps

Our past and ongoing work building applications for Colmena has clarified a few routes for future development.

Templates for Common Patterns

We developed Colmena with the goal of giving application designers the freedom to write any steering policy, but we found many common elements that were repeated across applications. One common example is an agent that submits the top task from a priority queue paired with an agent that updates the priority queue based on other completed computations. Providing a library of templates will accelerate application development while ensuring access to well-tuned implementations of scheduling patterns.

Integration with Model Repositories

Machine learning approaches for even well-established problems are far from stagnant, which means the AI components of applications will be continually refreshed. We plan to integrate Colmena with machine learning model repositories such as Hugging Face (Hugging Face 2023) and Garden (Garden Project 2023) so that models can be treated as interchangeable components rather than hard-coded elements within an application.

Intelligent Initialization

The cost of reloading machine learning models has been a consistent challenge in developing Colmena applications, yet one we have only addressed with ad-hoc solutions. A next step in Colmena development, in tandem with our workflow engine partners, will be to develop “model registries” or stateful actors that persist a shared state on workers between invocations of the same task. Our initial prototype also provides mechanisms to define routes for clearing stateful objects as they become unneeded (Brace and Pauloski 2023).

Streaming Intermediate Results

Running some tasks as persistent services, rather than discrete tasks, will provide many advantages including bypassing startup costs and dividing the costs of data transfer. Introducing services will require breaking the

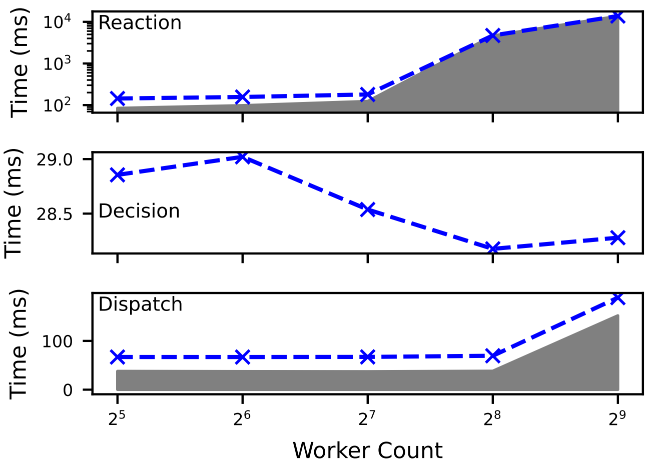

Figure 7. Latencies measured for a proxy application on up to 8 nodes of ALCF's Polaris supercomputer with task data sizes of 10 MB, a mean task length of 10 s, and a task length variance of 1 s. Each subplot shows a latency measure of particular relevance to dynamic workflows: Reaction: Task completion communicated from compute node to steering process; Decision: Steering process decides the next task; Dispatch: New task delivered to idle compute node. In each case, the dashed line indicates the total latency and the shaded region the latency corresponding to latency without data transfer.

assumption of workflow engines that functions are pure, so a current and future aspect of research in Colmena is extending our programming model to support generator tasks that yield results continually without returning. To achieve the required performance for deploying AI at larger scales, the generator tasks will need to be integrated within the data fabric (i.e., ProxyStore) as well.

Proxy Application for Dynamic Workflows

The core challenge of dynamic workflows, in our experience, is the ability to rapidly respond to the completion of tasks with new tasks. The latency comes in three parts: a *reaction* time between when a computation completes to when the Thinker is notified, a *decision* time to produce the next time, and a *dispatch* time for the new task to be delivered to a compute node. We propose a single proxy application to determine the maximum scaling of a workflow system.

Our proxy application attempts to maintain a constant amount of tasks in the workflow. The Thinker starts by preparing a list of computations then launching exactly as many as available workers, then launching a new task as soon as another is completed until the original queue of work is exhausted. The tasks take an empty array as input, sleep for a duration drawn from a normal distribution, then return a random byte string. The task rate of the workflow is varied by changing the worker numbers and the sleep duration distribution, and the data communication costs are varied by altering the size of of the input and output data.

Figure 7 shows the performance of our Proxy application on ALCF's Polaris supercomputer. We find that the major limit to applications for Colmena is latency of reacting to completed tasks, which becomes large after 256 tasks of approximately 10s each with data sizes of 10MB – an approximate task rate of 25 tasks/second. The fact that the latency increases with worker count helps identify a maximum sustainable task rate given hardware and data sizes, which can be used to guide application design. The proxy application narrows down that our performance could benefit from further tuning of data fabric employed ProxyStore, or multiprocessing for processing completed tasks in Colmena.

As of Colmena v0.6.1, the "task-limit" applications are available as demonstration code in our GitHub repository.

Conclusions

We reviewed the inspiration, implementation, and impact of Colmena, a tool we designed as part of the Exascale Computing Project to explore routes for integrating AI into computational workflows on supercomputing systems. Colmena allows scientists to describe workflow execution policies as Python functions that schedule computations and data transfer on an HPC system. We have employed Colmena successfully to implement applications that engage different types of machine learning (supervised, generative, unsupervised) in a variety of scientific domains—work that has both identified broadly applicable strategies for combining AI and simulation and suggested priorities for further algorithmic and systems research. We intend to continue simplifying the creation of new Colmena applications while exploring more routes for tailoring workflows to best use AI.

Acknowledgements

LW, GP, RC, RT, and IF acknowledge support by the ExaLearn Co-design Center (Alexander et al. 2021) of the Exascale Computing Project (17-SC-20-SC) (Alexander et al. 2020), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, to develop Colmena and evaluate its performance on HPC. YB and KC were supported to integrate Parsl with Colmena by NSF Grant 1550588 and the ExaWorks Project within the Exascale Computing Project. GP, VHS, and KC were supported to develop ProxyStore by NSF Grant 2004894. This research used resources of the Argonne Leadership Computing Facility (ALCF), a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357, including via the ALCF Data Science Program. It also used resources provided by the University of Chicago's Research Computing Center.

References

Alexander F, Almgren A, Bell J, Bhattacharjee A, Chen J, Colella P, Daniel D, DeSlippe J, Diachin L, Draeger E, Dubey A, Dunning T, Evans T, Foster I, Francois M, Germann T, Gordon M, Habib S, Halappanavar M, Hamilton S, Hart W, Huang Z, Hungerford A, Kasen D, Kent PRC, Kolev T, Kothe DB, Kronfeld A, Luo Y, Mackenzie P, McCallen D, Messer B, Mniszewski S, Oehmen C, Perazzo A, Perez D, Richards D, Rider WJ, Rieben R, Roche K, Siegel A, Sprague M, Steefel C, Stevens R, Syamlal M, Taylor M, Turner J, Vay JL, Voter AF, Windus TL and Yelick K (2020) Exascale applications: Skin in the game. *Philosophical Transactions of the Royal Society A* 378(2166): 20190056.

Alexander FJ, Ang J, Bilbrey JA, Balewski J, Casey T, Chard R, Choi J, Choudhury S, Debusschere B, DeGennaro AM, Dryden N, Ellis JA, Foster I, Cardona CG, Ghosh S, Harrington P, Huang Y, Jha S, Johnston T, Kagawa A, Kannan R, Kumar N, Liu Z, Maruyama N, Matsuoka S, McCarthy E, Mohd-Yusof J, Nugent P, Oyama Y, Proffen T, Pugmire D, Rajamanickam S, Ramakrishniah V, Schram M, Seal SK, Sivaraman G, Sweeney C, Tan L, Thakur R, Van Essen B, Ward L, Welch P, Wolf M, Xantheas SS, Yager KG, Yoo S and Yoon BJ (2021) Co-design center for exascale machine learning technologies (ExaLearn). *The International Journal of High Performance Computing Applications* 35(6): 598–616.

Alsaadi A, Ward L, Merzky A, Chard K, Foster I, Jha S and Turilli M (2022) RADICAL-Pilot and Parsl: Executing heterogeneous workflows on HPC platforms. In: *IEEE/ACM Workshop on Workflows in Support of Large-Scale Science*. IEEE. DOI:10.1109/works56498.2022.00009. URL http://dx.doi.org/10.1109/WORKS56498.2022.00009.

Babuji Y, Woodard A, Li Z, Clifford B, Kumar R, Lacinski L, Chard R, Wozniak J, Foster I, Wilde M, Katz D and Chard K (2019) Parsl: Pervasive parallel programming in Python. In: *ACM International Symposium on High-Performance Parallel and Distributed Computing*.

Badra J, Owoyele O, Pal P and Som S (2022) A machine learning-genetic algorithm approach for rapid optimization of internal combustion engines. In: *Artificial Intelligence and Data Driven Optimization of Internal Combustion Engines*. Elsevier, p. 125–158. DOI:10.1016/b978-0-323-88457-0.00003-5. URL http://dx.doi.org/10.1016/B978-0-323-88457-0.00003-5.

Balaprakash P, Salim M, Uram TD, Vishwanath V and Wild SM (2018) DeepHyper: Asynchronous hyperparameter search for deep neural networks. In: *2018 IEEE 25th international conference on high performance computing (HiPC)*. IEEE, pp. 42–51.

Bhowmik D, Gao S, Young MT and Ramanathan A (2018) Deep clustering of protein folding simulations. *BMC bioinformatics* 19: 47–58.

Brace A and Pauloski JG (2023) https://github.com/braceal/parsl_object_registry. Accessed: 2024-02-13.

Brace A, Salim M, Subbiah V, Ma H, Emani M, Trifa A, Clyde AR, Adams C, Uram T, Yoo H et al. (2021) Stream-AI-MD: Streaming AI-driven adaptive molecular simulations for heterogeneous computing platforms. In: *Proceedings of the Platform for Advanced Scientific Computing Conference*. pp. 1–13.

Brace A, Yakushin I, Ma H, Trifan A, Munson T, Foster I, Ramanathan A, Lee H, Turilli M and Jha S (2022) Coupling streaming AI and HPC ensembles to achieve 100–1000× faster biomolecular simulations. In: *IEEE International Parallel and Distributed Processing Symposium*. IEEE, pp. 806–816.

Caccin M, Li Z, Kermode JR and De Vita A (2015) A framework for machine-learning-augmented multiscale atomistic simulations on parallel supercomputers. *International Journal of Quantum Chemistry* 115(16): 1129–1139. DOI:10.1002/qua.24952. URL http://dx.doi.org/10.1002/qua.24952.

Carbone P, Katsifodimos A, Ewen S, Markl V, Haridi S and Tzoumas K (2015) Apache Flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering* 38(4).

Carriero N and Gelernter D (1988) Applications experience with Linda. *ACM SIGPLAN Notices* 23(9): 173–187.

Casalino L, Dommer AC, Gaieb Z, Barros EP, Sztain T, Ahn SH, Trifan A, Brace A, Bogetti AT, Clyde A et al. (2021) AI-driven multiscale simulations illuminate mechanisms of SARS-CoV-2 spike dynamics. *The International Journal of High Performance Computing Applications* 35(5): 432–451.

Chard R, Babuji Y, Li Z, Skluzacek T, Woodard A, Blaiszik B, Foster I and Chard K (2020) funcX: A federated function serving fabric for science. In: *29th Intl Symp on High-Performance Parallel Dist Computing*.

Curtarolo S, Morgan D, Persson K, Rodgers J and Ceder G (2003) Predicting crystal structures with data mining of quantum calculations. *Physical Review Letters* 91(13). DOI:10.1103/physrevlett.91.135503. URL http://dx.doi.org/10.1103/PhysRevLett.91.135503.

Dhakal A, Raith P, Ward L, Hong Enriquez RP, Rattihalli G, Chard K, Foster I and Milojicic D (2023) Fine-grained accelerator partitioning for machine learning and scientific computing in function as a service platform. In: *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W 2023. ACM. DOI:10.1145/3624062.3624238. URL http://dx.doi.org/10.1145/3624062.3624238.

Dharuman G, Ward L, Ma H, Setty PV, Gokdemir O, Foreman S, Emani M, Hippe K, Brace A, Keipert K, Gibbs T, Foster I, Anandkumar A, Vishwanath V and Ramanathan A (2023) Protein generation via genome-scale language models with bio-physical scoring. In: *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, SC-W 2023. ACM. DOI:10.1145/3624062.3626087. URL http://dx.doi.org/10.1145/3624062.3626087.

Doan HA, Agarwal G, Qian H, Counihan MJ, Rodríguez-López J, Moore JS and Assary RS (2020) Quantum chemistry-informed active learning to accelerate the design and discovery of sustainable energy storage materials. *Chemistry of Materials* 32(15): 6338–6346. URL https://doi.org/10.1021/acs.chemmater.0c00768.

Dommer A, Casalino L, Kearns F, Rosenfeld M, Wauer N, Ahn SH, Russo J, Oliveira S, Morris C, Bogetti A et al. (2023) # COVIDisAirborne: AI-enabled multiscale computational microscopy of delta SARS-CoV-2 in a respiratory aerosol. *The International Journal of High Performance Computing Applications* 37(1): 28–44.

Dror RO, Dirks RM, Grossman J, Xu H and Shaw DE (2012) Biomolecular simulation: A computational microscope for molecular biology. *Annual review of biophysics* 41: 429–452.

Dunn A, Brenneck J and Jain A (2019) Rocketsled: A software library for optimizing high-throughput computational searches. *J Physics: Materials* 2(3): 034002.

Egele R, Maulik R, Raghavan K, Lusch B, Guyon I and Balaprakash P (2022) AutoDEUQ: Automated deep ensemble with uncertainty quantification. In: *26th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 1908–1914.

Ejarque J, Badia RM, Albertin L, Aloisio G, Baglione E, Becerra Y, Boschert S, Berlin JR, D'Anca A, Elia D, Exertier F, Fiore S, Flich J, Folch A, Gibbons SJ, Koldunov N, Lordan F, Lorito S, Løvholt F, Macías J and Volpe M (2022) Enabling dynamic and intelligent workflows for HPC, data analytics, and AI convergence. *Future Generation Computer Systems* 134: 414–429.

Ferreira da Silva R, Badia RM, Bard D, Foster IT, Jha S and Suter F (2024) Frontiers in scientific workflows: Pervasive integration with high-performance computing. *Computer* 57(8): 36–44. DOI:10.1109/mc.2024.3401542. URL `http://dx.doi.org/10.1109/MC.2024.3401542`.

Foster I and Taylor S (1989) *Strand: New Concepts in Parallel Programming*. Prentice-Hall, Inc.

Fox G, Adiga A, Chen J, Beckstein O, Jha S, Glazier JA, Kadupitiya J, Jadhao V, Kim M, Qiu J, Sluka JP, Somogyi E and Marathe M (2019) Learning everywhere: Pervasive machine learning for effective high-performance computation. In: *IEEE International Parallel and Distributed Processing Symposium Workshops*.

Frey J, Tannenbaum T, Livny M, Foster I and Tuecke S (2002) Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing* 5: 237–246.

Garden Project (2023) `https://garden-ai.readthedocs.io/`. Accessed: 2024-02-13.

Gómez-Bombarelli R, Wei JN, Duvenaud D, Hernández-Lobato JM, Sánchez-Lengeling B, Sheberla D, Aguilera-Iparraguirre J, Hirzel TD, Adams RP and Aspuru-Guzik A (2018) Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science* 4(2): 268–276. DOI:10.1021/acscentsci.7b00572. URL `http://dx.doi.org/10.1021/acscentsci.7b00572`.

Harb H, Elliott SN, Ward L, Foster IT, Klippenstein SJ, Curtiss LA and Assary RS (2023) Uncovering novel liquid organic hydrogen carriers: A systematic exploration of chemical compound space using cheminformatics and quantum chemical methods. *Digital Discovery* 2(6): 1813–1830. DOI:10.1039/d3dd00123g. URL `http://dx.doi.org/10.1039/D3DD00123G`.

Herath C and Plale B (2010) Streamflow programming model for data streaming in scientific workflows. In: *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. DOI:10.1109/ccgrid.2010.116.

Hospital A, Goñi JR, Orozco M and Gelpí JL (2015) Molecular dynamics simulations: Advances and applications. *Advances and Applications in Bioinformatics and Chemistry* : 37–47.

Hudson S, Larson J, Navarro JL and Wild SM (2022) libEnsemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations. *IEEE Transactions on Parallel and Distributed Systems* 33(4): 977–988. DOI:10.1109/tpds.2021.3082815.

Huerta E, Park H, Yan X, Zhu R, Chaudhuri S, Cooper D, Foster I and Tajkhorshid E (2023) GHP-MOFassemble: Diffusion modeling, high throughput screening, and molecular dynamics for rational discovery of novel metal-organic frameworks for carbon capture at scale. DOI:10.21203/rs.3.rs-3084157/v1. URL `http://dx.doi.org/10.21203/rs.3.rs-3084157/v1`.

Hugging Face (2023) `https://huggingface.co/`. Accessed: 2024-02-13.

Jacobsen T, Jørgensen M and Hammer B (2018) On-the-fly machine learning of atomic potential in density functional theory structure optimization. *Physical Review Letters* 120(2). DOI:10.1103/physrevlett.120.026102. URL `http://dx.doi.org/10.1103/PhysRevLett.120.026102`.

Kamatar A, Sakarvadia M, Hayot-Sasson V, Chard K and Foster I (2023) Lazy Python dependency management in large-scale systems. In: *IEEE 19th International Conference on e-Science*. DOI:10.1109/e-science58273.2023.10254910.

Lee H, Turilli M, Jha S, Bhowmik D, Ma H and Ramanathan A (2019) DeepDriveMD: Deep-learning driven adaptive molecular simulations for protein folding. In: *3rd Workshop on Deep Learning on Supercomputers*.

Lin Z, Akin H, Rao R, Hie B, Zhu Z, Lu W, Smetanin N, Verkuil R, Kabeli O, Shmueli Y, dos Santos Costa A, Fazel-Zarandi M, Sercu T, Candido S and Rives A (2023) Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science* 379(6637): 1123–1130. DOI:10.1126/science.ade2574. URL `http://dx.doi.org/10.1126/science.ade2574`.

Liu J, Pacitti E, Valduriez P and Mattoso M (2015) A survey of data-intensive scientific workflow management. *Journal of Grid Computing* 13(4): 457–493. DOI:10.1007/s10723-015-9329-8. URL `http://dx.doi.org/10.1007/s10723-015-9329-8`.

Montoya JH, Winther KT, Flores RA, Bligaard T, Hummelshøj JS and Aykol M (2020) Autonomous intelligent agents for accelerated materials discovery. *Chemical Science* 11(32): 8517–8532. DOI:10.1039/d0sc01101k. URL `http://dx.doi.org/10.1039/D0SC01101K`.

Moritz P, Nishihara R, Wang S, Tumanov A, Liaw R, Liang E, Elibol M, Yang Z, Paul W, Jordan MI and Stoica I (2018) Ray: A distributed framework for emerging AI applications. In: *13th USENIX OSDI*. ISBN 978-1-939133-08-3, pp. 561–577.

Partee S, Ellis M, Rigazzi A, Shao AE, Bachman S, Marques G and Robbins B (2022) Using machine learning at scale in numerical simulations with SmartSim: An application to ocean climate modeling. *Journal of Computational Science* 62: 101707. DOI:https://doi.org/10.1016/j.jocs.2022.101707. URL `https://www.sciencedirect.com/science/article/pii/S1877750322001065`.

Pauloski JG, Hayot-Sasson V, Ward L, Brace A, Bauer A, Chard K and Foster I (2024) Object Proxy Patterns for Accelerating Distributed Applications. URL `https://arxiv.org/abs/2407.01764`.

Pauloski JG, Hayot-Sasson V, Ward L, Hudson N, Sabino C, Baughman M, Chard K and Foster I (2023) Accelerating communications in federated applications with transparent object proxies. In: *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23. DOI:10.1145/3581784.3607047.

Phillips JC, Zheng G, Kumar S and Kalé LV (2002) NAMD: Biomolecular simulation on thousands of processors. In: *ACM/IEEE Conference on Supercomputing*. IEEE, pp. 36–36.

Phung TS, Ward L, Chard K and Thain D (2021) Not all tasks are created equal: Adaptive resource allocation for heterogeneous tasks in dynamic workflows. In: *IEEE Workshop on Workflows in Support of Large-Scale Science*. DOI:10.1109/works54523.2021.00008. URL `http://dx.doi.org/10.1109/WORKS54523.2021.00008`.

Reyes KG, Liu J and Vargas CJD (2022) Decision-making under uncertainty for multi-stage pipelines: Simulation studies to benchmark screening strategies. *JOM* 74(8): 2897–2907. DOI:10.1007/s11837-022-05368-z. URL `http://dx.doi.org/10.1007/s11837-022-05368-z`.

Russo JD, Zhang S, Leung JM, Bogetti AT, Thompson JP, DeGrave AJ, Torrillo PA, Pratt A, Wong KF, Xia J, Wong KF, Xia J, Copperman J, Adelman JL, Zwier MC, LeBard DN, Zuckerman DM and Chong LT (2022) WESTPA 2.0: High-performance upgrades for weighted ensemble simulations and analysis of longer-timescale applications. *Journal of Chemical Theory and Computation* 18(2): 638–649.

Shamsi Z, Cheng KJ and Shukla D (2018) Reinforcement learning based adaptive sampling: REAPing rewards by exploring protein conformational landscapes. *The Journal of Physical Chemistry B* 122(35): 8386–8395.

St John P, Phillips C, Kemper TW, Wilson AN, Guan Y, Crowley MF, Nimlos MR and Larsen RE (2019) Message-passing neural networks for high-throughput polymer screening. *J Chemical Physics* 150(23).

Thain D, Tannenbaum T and Livny M (2005) Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience* 17(2-4): 323–356.

Trifan A, Gorgun D, Salim M, Li Z, Brace A, Zvyagin M, Ma H, Clyde A, Clark D, Hardy DJ et al. (2022) Intelligent resolution: Integrating Cryo-EM with AI-driven multi-resolution simulations to observe the severe acute respiratory syndrome coronavirus-2 replication-transcription machinery in action. *The International Journal of High Performance Computing Applications* 36(5-6): 603–623.

Ward L, Pauloski JG, Hayot-Sasson V, Chard R, Babuji Y, Sivaraman G, Choudhury S, Chard K, Thakur R and Foster I (2023) Cloud services enable efficient AI-guided simulation workflows across heterogeneous resources. In: *IEEE International Parallel and Distributed Processing Symposium Workshops*. DOI:10.1109/ipdpsw59300.2023.00018. URL http://dx.doi.org/10.1109/IPDPSW59300.2023.00018.

Ward L, Sivaraman G, Pauloski JG, Babuji Y, Chard R, Dandu N, Redfern PC, Assary RS, Chard K, Curtiss LA, Thakur R and Foster I (2021) Colmena: Scalable machine-learning-based steering of ensemble simulations for high performance computing. In: *IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments*. IEEE. DOI: 10.1109/mlhpc54614.2021.00007. URL http://dx.doi.org/10.1109/MLHPC54614.2021.00007.

Wilde M, Foster I, Iskra K, Beckman P, Zhang Z, Espinosa A, Hategan M, Clifford B and Raicu I (2009) Parallel scripting for applications at the petascale and beyond. *Computer* 42(11): 50–60.

Woo HM, Qian X, Tan L, Jha S, Alexander FJ, Dougherty ER and Yoon BJ (2023) Optimal decision-making in high-throughput virtual screening pipelines. *Patterns* 4(11): 100875. DOI:10.1016/j.patter.2023.100875. URL http://dx.doi.org/10.1016/j.patter.2023.100875.

Wozniak JM, Jain R, Balaprakash P, Ozik J, Collier NT, Bauer J, Xia F, Brettin T, Stevens R, Mohd-Yusof J, Cardona CG, Essen BV and Baughman M (2018) CANDLE/Supervisor: A workflow framework for machine learning applied to cancer research. *BMC Bioinformatics* 19(S18).

Yildiz O, Morozov D, Nicolae B and Peterka T (2021) Dynamic heterogeneous task specification and execution for in situ workflows. In: *Workshop on Workflows in Support of Large-Scale Science*.

Zamora Y, Ward L, Sivaraman G, Foster I and Hoffmann H (2021) Proxima: Accelerating the integration of machine learning in atomistic simulations. In: *ACM International Conference on Supercomputing*, ICS '21. ACM. DOI:10.1145/3447818.3460370. URL http://dx.doi.org/10.1145/3447818.3460370.

Zhang Y, Wang H, Chen W, Zeng J, Zhang L, Wang H and E W (2020) DP-GEN: A concurrent learning platform for the generation of reliable deep learning based potential energy models. *Computer Physics Communications* 253: 107206. DOI:10.1016/j.cpc.2020.107206. URL http://dx.doi.org/10.1016/j.cpc.2020.107206.

Zhao Y, Dobson J, Foster I, Moreau L and Wilde M (2005) A notation and system for expressing and executing cleanly typed workflows on messy scientific data. *ACM SIGMOD Record* 34(3): 37–43.

Zvyagin M, Brace A, Hippe K, Deng Y, Zhang B, Bohorquez CO, Clyde A, Kale B, Perez-Rivera D, Ma H, Mann CM, Irvin M, Ozgulbas DG, Vassilieva N, Pauloski JG, Ward L, Hayot-Sasson V, Emani M, Foreman S, Xie Z, Lin D, Shukla M, Nie W, Romero J, Dallago C, Vahdat A, Xiao C, Gibbs T, Foster I, Davis JJ, Papka ME, Brettin T, Stevens R, Anandkumar A, Vishwanath V and Ramanathan A (2023) GenSLMs: Genome-scale language models reveal SARS-CoV-2 evolutionary dynamics. *The International Journal of High Performance Computing Applications* 37(6): 683–705. DOI: 10.1177/10943420231201154. URL http://dx.doi.org/10.1177/10943420231201154.