PDF Download
3731599.3767350.pdf
24 December 2025
Total Citations: 0
Total Downloads: 1163

Published: 16 November 2025

Citation in BibTeX format

SC Workshops '25: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis
November 16 - 21, 2025
MO, St Louis, USA

Conference Sponsors:
SIGHPC

RESEARCH-ARTICLE

A Query Engine for Scientific Data Exploration using Theory, Simulation, and Artificial Intelligence Models

ABHISHEK DWARAKI, Hewlett-Packard Inc., Palo Alto, CA, United States

SREENIVAS RANGAN SUKUMAR, Hewlett Packard Enterprise, Palo Alto, CA, United States

CHRISTOPHER D RICKETT, Hewlett Packard Enterprise, Palo Alto, CA, United States

CLARETE RIANA CRASTA, Hewlett Packard Enterprise, Palo Alto, CA, United States

HARUMI ANNE KUNO, Hewlett Packard Enterprise, Palo Alto, CA, United States

MICHAEL NEAL, University of Mississippi School of Pharmacy, University, MS, United States

View all

A Query Engine for Scientific Data Exploration using Theory, Simulation, and Artificial Intelligence Models

Abhishek Dwaraki[*] [†]
Hewlett Packard Enterprise (HPE)
Ft. Collins, CO, USA
abhishek.dwaraki@gmail.com

Sreenivas. R. Sukumar
Hewlett Packard Enterprise (HPE)
Seattle, WA, USA
sreenivas.sukumar@hpe.com

Christopher D. Rickett
Hewlett Packard Enterprise (HPE)
Indianapolis, IN, USA
chris.rickett@hpe.com

Clarete Riana Crasta
Hewlett Packard Enterprise (HPE)
Poughkeepsie, NY, USA
clarete.riana@hpe.com

Harumi Kuno
Hewlett Packard Enterprise (HPE)
Milpitas, CA, USA
harumi.kuno@hpe.com

Michael Neal[‡]
National Center for Natural Products Research
University of Mississippi
Oxford, MS, USA
wneal@olemiss.edu

Pankaj Pandey
National Center for Natural Products Research
University of Mississippi
Oxford, MS, USA
ppandey@olemiss.edu

Ryan Yates
National Center for Natural Products Research
University of Mississippi
Oxford, MS, USA
cryates2@olemiss.edu

Karlon West
Hewlett Packard Enterprise (HPE)
Spring, TX, USA
karlon.west@hpe.com

Amar Gopal Chittiboyina
National Center for Natural Products Research
University of Mississippi
Oxford, MS, USA
amar@olemiss.edu

Ikhlas A. Khan
National Center for Natural Products Research
University of Mississippi
Oxford, MS, USA
ikhan@olemiss.edu

John L. Byrne
Hewlett Packard Enterprise (HPE)
Pasadena, CA, USA
john.l.byrne@hpe.com

Sekwon Lee
Hewlett Packard Enterprise (HPE)
Austin, TX, USA
sekwon.lee@hpe.com

David Emberson
Hewlett Packard Enterprise (HPE)
Milpitas, CA, USA
emberson@hpe.com

Abstract

Modern scientific discovery increasingly couples large-scale simulations, heterogeneous data, and AI models into interactive, iterative workflows. Yet existing systems rarely let scientists compose expressive queries that both retrieve massive, multi-modal datasets and invoke complex computational models (simulations or AI inferences) with low turnaround time. We present the Intelligent Data Search (IDS) framework to bridge this gap. IDS builds on the Cray Graph Engine and combines a scalable in-memory datastore (acting as a feature store, vector store, and knowledge graph) with a unified query engine that spans keyword, set-theoretic, and linear-algebraic operators. It also incorporates a model repository for User-Defined Functions (UDFs) and pre-trained AI models, and a globally distributed, multi-tier cache to stash intermediate and simulation outputs. We demonstrate IDS on a life-sciences workflow developed in collaboration with the National Center for Natural Products (NCNPR). This effort integrates AI models such as AlphaFold with simulation codes like AutoDock Vina, and analytic functions such as the Smith–Waterman algorithm within a single query. In our evaluation, we show strong scaling on HPC systems, a complex "what-could-be" query that executes millions of similarity searches and thousands of model inferences. This executes in tens of seconds at scale, resulting in a 5–15x end-to-end improvement from the distributed cache. IDS enables scientists to pose and iterate model-driven "what-if" questions over petascale datasets

[*]Chris Rickett and Abhishek Dwaraki both contributed equally to this research
[†]Hewlett Packard Enterprise
[‡]University of Mississippi

SC Workshops '25, St Louis, MO, USA
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767350

with far lower latency and less workflow fragmentation than prior approaches.

CCS Concepts

• **Computing methodologies** → **Simulation tools**; **Molecular simulation**; • **Information systems** → *Query representation.*

Keywords

molecular-docking, hpc, simulations, computational-biology

ACM Reference Format:
Abhishek Dwaraki, Sreenivas. R. Sukumar, Christopher D. Rickett, Clarete Riana Crasta, Harumi Kuno, Michael Neal, Pankaj Pandey, Ryan Yates, Karlon West, Amar Gopal Chittiboyina, Ikhlas A. Khan, John L. Byrne, Sekwon Lee, and David Emberson. 2025. A Query Engine for Scientific Data Exploration using Theory, Simulation, and Artificial Intelligence Models. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3731599.3767350

1 Introduction

The synergy of high-performance computing (HPC) and artificial intelligence (AI) is fundamentally reshaping scientific discovery. Modern scientific workflows are evolving beyond simple data processing to become complex, dynamic tasks that integrate large-scale simulations, computational modeling, and machine learning inferences. This paradigm shift is particularly evident in fields such as drug discovery, where the interactive exploration of molecular structures and properties can drastically accelerate hypothesis generation and reduce the time and cost associated with screening novel compounds. However, this new mode of scientific exploration presents a significant challenge: how to efficiently manage, query, and analyze massive, often multi-modal, datasets in a manner that is both intuitive and performant for scientists. For example, current scientific discovery workflows require scientists to manually orchestrate complex pipelines that combine: 1. petabyte-scale data queries across heterogeneous formats, 2. computationally expensive simulations (minutes to hours per calculation), 3. AI model inference on intermediate results, and 4. Interactive exploration with sub-second response requirements.

We identify two practical obstacles for interactive scientific exploration: (i) limited expressiveness — scientists cannot easily ask "what-if" questions that require chaining queries with model executions, and (ii) cumulative latency — repeated or overlapping model runs create large intermediate-result sets and long turn-around times at scale. This fragmentation prevents scientists from asking integrated questions like "Find all protein structures similar to X, predict their binding affinities using model Y, and run molecular dynamics on the top 10 candidates" as a single, optimized query.

We focus on the Intelligent Data Search (IDS) framework, which addresses both axes, enabling expressive, model-driven queries and making those queries performant on HPC-scale datasets. Built on the Cray Graph Engine (CGE), IDS's architecture is specifically tailored for HPC environments, featuring a scalable in-memory database, a versatile query engine, and a globally distributed cache. At its core, the IDS in-memory datastore leverages HPC principles for the optimal management and utilization of unstructured data. This datastore functions as a 3-in-1 feature store, vector store, and knowledge graph host, capable of accommodating a wide variety of data types, including documents, 2D images, 3D point clouds, genomic sequences, and vector embeddings. This allows unified query semantics across modalities, co-locating multiple critical functionalities in a single framework; scientists can query extensive datasets with a unified approach that integrates keyword search, set-theoretic operations, and linear-algebraic methods. Beyond simple data retrieval, IDS includes a repository of computational models, spanning domain-specific algorithms, open-source software, pre-trained AI models, and traditional HPC simulation codes. The intelligent query planner orchestrates complex queries that may require executing both traditional HPC simulations and rapid AI inferences.

This capability allows for the use of sophisticated user-defined functions, such as custom similarity metrics and pattern searches powered by AI models. In this paper, the practical utility of the IDS framework is demonstrated on a life sciences use case developed in collaboration with the National Center for Natural Products Research (NCNPR). This work integrates AI models like AlphaFold for structure prediction, MolGAN for molecular generation, HPC codes like Autodock Vina for molecular docking and the Smith-Waterman (SW) algorithm for protein similarity assessment in a workflow. By empowering scientists to ask sophisticated questions, IDS effectively addresses the "what-is," "what-else," "what-if," and "what-could-be" facets of the discovery process. A simple "what-is" query returns in milliseconds. A complex "what-could-be" query—executing millions of similarity searches, thousands of AI inferences, and HPC simulations over a 100B-node medical graph ($\approx$ 30 TB)—completes in $\approx$ 60 seconds using 256 nodes. These are actual measurements captured in our previous evaluations.

A key improvement presented in this paper is the resolution of a critical bottleneck: the performance and scalability of a workflow. We address this by introducing a globally distributed, cluster-wide cache designed to accelerate computations. This multi-tiered and queryable cache allows another IDS instance on the same cluster to access and reuse results from prior simulations and queries, thereby significantly reducing computational latency and enhancing overall efficiency. Our primary contributions are summarized as follows:

- The development of the IDS platform, a novel framework for scientific data search that enables expressive, model-driven queries,
- A domain-agnostic demonstration of scientific data search capabilities, showcasing its versatility across various research domains,
- Strong scaling to large data sizes and complex queries, demonstrating performance at scale, and
- The implementation of an optimal query planner and a globally distributed, client-side cache designed for HPC clusters.

The rest of this paper is organized as follows: Section 2 provides a detailed background of the intelligent search framework, focusing on the query engine, model repository, and query planner. Section 3 then delves into the design and architecture of the global, distributed cache and explains its role in the overall system. In Section 4, we illustrate the practical application of this powerful computational framework by solving real-world problems. Section 5 outlines our experimental design and evaluation methodology before presenting an analysis of the results. We discuss similar approaches and the current state-of-the-art in Section 6. Finally, we offer concluding remarks and outline future work in Sections 7 and 8.

2 Intelligent Data Search

2.1 Background

The IDS framework is a scalable, massively parallel processing database for unstructured data. It is built upon the Cray Graph Engine (CGE), a well-established semantic graph database whose performance in hosting knowledge graphs and performing analytics has been validated in peer-reviewed literature. The foundations of CGE can be traced to the Urika-GD database appliance, first launched in 2012. This initial implementation was designed for the Cray XMT2 shared-memory machine, utilizing a multi-threaded programming model to achieve efficient graph searches and hide memory latency. The technology was later ported to the XC30 architecture in 2015, leveraging the Partitioned Global Address Space (PGAS) programming model and Coarray-C++ [27]. To enhance portability and performance across diverse platforms, subsequent development efforts replaced the Coarray-C++ and PGAS libraries with versions built on top of MPI and POSIX shared memory. This allowed CGE to run optimally on a variety of systems, including HPE Cray EX supercomputers, HPE SuperDome Flex shared-memory machines, parallel clusters, and cloud deployments [39]. Today, IDS utilizes CGE's capabilities to accelerate data ingestion and query execution with GPUs. This enables the platform to handle computationally complex queries and deliver high-quality results with reduced latency.

2.2 Architecture of the IDS Framework

The IDS framework is designed as a massively parallel data search platform with the following core objectives:

- Data Management: Store, manage, host, and process multi-modal data represented as knowledge graphs.
- Query Capabilities: Provide interactive query and semantic traversal for data-driven discovery.
- Algorithmic Acceleration: Accelerate domain-specific user-defined functions (UDFs)
  and graph algorithms such as PageRank.
- Workflow Execution: Execute query workflows across multiple datasets to generate hypotheses.

The architecture simplifies the creation of query workflows that combine knowledge graphs with domain-specific UDFs, AI operations, and HPC-style simulations. Figure 1 illustrates the architecture of our solution.

The IDS framework has the following components:

- the *Datastore Launcher* handles user operations such as launch, opening the query/update endpoint and tear down,
- a *Datastore Client* submits queries/updates, fetches logs, and adds new user codes,

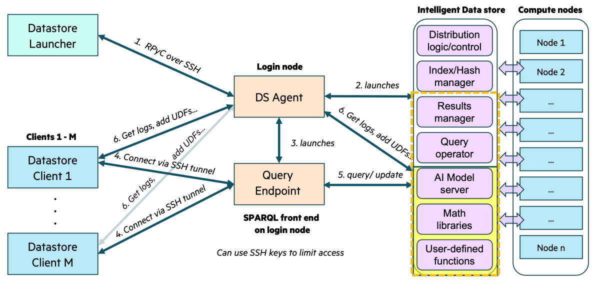


Figure 1: Architecture of the Intelligent Data Search Framework

- a *Datastore Agent* lives on the compute nodes, and works in conjunction with the DataStore Launcher and the DataStore-Client to launch/tear down the framework, create connections to the query endpoints, retrieve logs, and import new user codes, and
- the *IDS backend*, built on top of CGE, is the massively parallel graph engine executing queries on the compute node(s).

The IDS framework is designed for scalability and can be deployed across a variety of hardware environments, from large-scale clusters like the Superdome Flex and ProLiant servers to public-cloud virtual machines and even individual laptops. The framework's ability to package popular Python libraries as user-defined environments enables easy access to data science tools that are essential for large-scale explorations directly from familiar environments like Jupyter notebooks. This versatility allows scientists to execute workflows consistently across different platforms, often interacting with the system's front end via Jupyter notebooks.

2.3 Bridging Graph Operations, User Defined Functions and Simulations

In this section, we discuss how IDS blends traditional graph operations with domain-specific UDFs and AI-powered UDFs to augment simulations at the HPC scale. The design of IDS is aimed at simplifying its use for data scientists by leveraging Python, the language commonly used for many AI applications. To further this goal, IDS offers a comprehensive Python client API that allows users to perform all necessary actions, such as launch or tear down, data ingestion, UDF additions, etc., from a Python script or Jupyter notebook. IDS builds on efforts that containerize CGE [39] by incorporating the IDS Python client code and other essential components into the container, ensuring optimal execution on the given hardware. This approach allows users to easily launch IDS on their laptop and then transition to a larger system using the same container.

IDS enables users to easily import their own code and execute it in conjunction with their queries by providing them with the requisite Python APIs to achieve this. CGE allows users to define their own functions, which could be compiled into a shared object that it would load at launch time [38]. This functionality was primarily designed for C/C++ programs and was static because they cannot be modified once IDS launched. For users that need to continuously incorporate new AI models or domain-specific functions, IDS includes an API that allows the user to dynamically load their Python UDFs. Since loading Python modules can be time-consuming, IDS maintains a cache of the loaded user modules to ensure the overhead is only incurred the first time a module loads. However, to allow users to make continual updates to their code and use it in running queries, IDS includes a special function that forces IDS to reload the module. These enhancements significantly increase IDS's flexibility, allowing users to execute a wider variety of queries and process different types of input data, either at load time or dynamically within queries. Consequently, this IDS API allows users to massively parallelize their workflows, backed by an extremely efficient graph datastore to accelerate their simulations.

In today's rapidly evolving technological landscape, AI is at the forefront of many advancements. However, UDFs that incorporate pre-trained AI models or simulation-style code can vary significantly in their computational requirements, making it challenging for users to plan workflows for optimal execution. While users can perform some query optimizations based on their knowledge of the UDFs, this task can quickly become overwhelming as the number of UDFs per query increases. Given the rapid publication of new AI models on platforms such as HuggingFace [1] and the ease with which users can dynamically import new codes into IDS, it is easy to imagine users interactively searching data to test new models without fully understanding the performance implications. Moreover, as users refine their sub-graphs through searches, the behavior of UDFs may change due to differences in the data.

2.4 Query Planning and Optimization for AI-based UDFs

2.4.1 Query Profiling. A key requirement for enabling effective query planning and optimization is to profile UDFs effectively. IDS tracks statically linked UDFs using their unique name and dynamically loaded UDFs using the Python module name and method name. The UDF profiling information is maintained by each rank and includes: (i) the number of times it has been executed, (ii) total UDF execution time, and finally (iii) the number of times a query expression was rejected due to the UDF's execution.

The UDF entry in the profiling data-store is continually updated through the life time of a running IDS instance. This is essential for optimizing queries dynamically over time as data may change and new UDFs may be added. Furthermore, rank based profiling better enables query optimizations since IDS can tailor these operations for a specific scenario or rank's requirement.

2.4.2 Solution Re-balancing. Solution re-balancing is one of the processes where UDF profiling information is utilized, specifically at the beginning of FILTER operations. IDS commonly re-balances solutions across ranks between operations (e.g., scans, joins, merges). However, this approach may not always be effective for UDF expressions. Execution times can vary across different ranks due to factors such as node hardware and differences in the sub-graph within each rank's data shard. Since these execution times are more indicative of the load a rank is handling, IDS incorporates these times along with solution counts when re-balancing.

Within IDS, expressions evaluated as part of operators (e.g., FILTER) are represented as expression trees. The FILTER operation optimizes expression trees that contain UDFs by leveraging the profiling information. Each rank estimates the time to evaluate a single solution and then all ranks exchange the estimates to determine an execution speed ratio, defined as the ratio between the slowest rank and all the others. If all ranks have a reasonably similar throughput, (within ∼ 20% of the slowest one), re-balancing defaults to query count-based re-balancing. However, when some ranks are considerably faster than others, the following process is used:

- Each rank estimates solutions per second
- Calculate ratio of each rank to slowest (*rank_ratio*)
- Sum number of solutions per second
- Divide number of solutions by solutions per second to create a (*chunk_size*)
- Redistribute by assigning each rank (*chunk_size*∗*rank_ratio*) solutions. For example:
  - Given 1.4 million intermediate solutions and 900 ranks with 500 ranks processing 100 ops/s, 300 processing 200 ops/s and 100 processing 300 ops/s:
    - ∗ 500 + (300 * 2) + (100 * 3) = 1400 solutions per second
    - ∗ 1.4M / 1.4K = 10K solutions per chunk
      - · 500 slowest ranks = 10K solutions
      - · 300 slightly quicker ranks = 20K solutions
      - · 100 fastest ranks = 30K solutions
- UDF evaluations are independent across ranks, so the overall time is bounded by the slowest rank. In the above example, this could be ∼10K/100 = 100 seconds whereas vanilla query count re-balancing results in an estimated time of ∼14K / 100 per second = 140 seconds.

In the next section, we discuss the method of optimizing expressions that target AI models for inferencing and other purposes.

2.4.3 Optimizing AI/ML Pipeline Expressions. User-defined functions are used heavily in AI/ML to filter intermediate search and inference results as the control flows through the pipelines. Before executing expressions that contain UDF calls as part of a FILTER operation, each rank in IDS uses the profiling information to perform time estimation. Each MPI rank will then traverse the expression tree to create sets of chained conditional operators. If a chain of conditional operators includes a UDF with profiling information, the rank will reorder the expressions to process them in ascending order of estimated evaluation time. If two UDFs have similar computational times, the function expected to eliminate more solutions is prioritized to reduce the number of evaluations for each intermediate solution.

Since the expression evaluations are carried out per-rank and are independent, the ranks will sync solutions globally only once the evaluations are complete. This allows ranks to tailor the reorderings to their resource availability, which could result in ranks evaluating the expression(s) in different orders.

3 Global Shared Client-Side Cache

Efficient storage and scalable caching are critical to enabling HPC and AI workloads that must store, manage, and serve very large datasets. As clusters grow, aggregate client-side resources (DRAM, NVMe, network) make sharing a client-side cache attractive when combined with RDMA-based access, smart placement, and good policies.

We implemented a globally shared, client-side cache (prototype described in [7]) that fronts distributed storage; the prototype focuses on DAOS [19, 23, 24] but exposes an API that can front other systems (e.g., Lustre), which we use in Section 5.

A client-side, globally shared cache scales with compute resources and reduces pressure on central storage by leveraging node-local memory and network bandwidth. Key challenges it addresses include: (i) supporting RDMA-optimized parallel applications, (ii) managing CPU/GPU memory transfers, and (iii) sharing cached data across parallel frameworks.

We drew design insights from related systems and projects [29, 42, 46, 48] to ensure the solution is robust and fits heterogeneous HPC/AI environments.

3.1 Architecture

Our goal is to allow applications to exploit memory and local SSDs across multiple nodes to cache and share data efficiently. This architecture is illustrated in Figure 2, where both compute nodes and dedicated memory server nodes are shown contributing memory to the cache. The memory contributions can come from various sources, including DRAM, local NVMe SSDs, and GPU memory. Each memory type forms a distinct tier within the cache. A distributed cache manager consolidates per-node resources into a unified cache, maintains awareness of the capacities and access latencies associated with each tier, and manages multiple tiers of the cache by implementing cache policies and data placement algorithms. Data is cached locally to the nodes where there is a higher probability of it being accessed, whenever possible, to enable efficient access patterns. This cache can also be shared across applications running in a cluster. DAOS operations in the applications are intercepted and cached as required in the client-side cache.

3.2 Cache Manager

The caching system is built to deliver efficient, scalable, and consistent data access in distributed environments. As depicted in Figure 3, at its core is the Cache Manager, which orchestrates cache operations, enforces policies, and manages metadata to ensure optimal performance. The cache manager plays a pivotal role in managing metadata across the distributed cache. By providing detailed data locality information, it enables frameworks and other components, such as schedulers, to effectively align computation with data partitions. This capability ensures data affinity, optimizing performance by reducing data movement and improving access efficiency in distributed environments. Additionally, the cache manager facilitates data movement between DRAM and SSDs based on user-provided hints or operator-defined policies, ensuring efficient utilization of resources and optimal performance for diverse workloads.

IDS caches the complete outputs of molecular docking simulations (Autodock Vina) as named cache objects. Each cached object is addressed by its object name/path and a computed object hash (object ID) (the TR-Cache C API exposes hash/ID helpers), and may be associated with registered buffers for RDMA-optimized transfers. The cache is a distributed, client-side layer whose metadata is managed by the Cache Manager while authoritative copies remain in persistent backing storage (e.g., DAOS). If a cache node fails its in-memory/SSD contents are lost but can be re-populated from the backing store when the node rejoins; the distributed metadata avoids a single point of failure.

The system also incorporates an RDMA-based data access framework, enabling high-speed, low-latency data movement across nodes, critical for distributed workloads. Furthermore, the caching system provides a comprehensive API and configuration interface, allowing seamless integration with applications, file systems, and object stores, while offering the flexibility to adapt to diverse workload requirements.

A portion of compute nodes' DRAM, memory nodes' DRAM, and optionally other memory types such as HBM, are allocated to form a globally distributed cache. This cache transparently supports RDMA-based high-speed data movement within and across applications, enabling low-latency access. When DRAM capacity is exceeded, the cache seamlessly spills data to locally connected SSDs, reducing overall DRAM requirements while maintaining performance. Additionally, the cache manager dynamically relocates data within the caching layer to optimize proximity to computation, leveraging user-defined hints or operator-defined policies to ensure efficient and cost-effective operation.

3.3 Remote Memory Access Using OpenFAM

The global cache provides a unified RDMA-based [31] framework and protocol for data sharing, offering minimal overhead while being dynamic, scalable, and adaptable to heterogeneous environments. To enable HPC workloads to access remote memory, the global cache leverages an open source project named OpenFAM [9, 40, 41], which provides a programming interface for building applications that leverage large-scale disaggregated memory. The OpenFAM API offers memory management and lightweight data operations, modelled after OpenSHMEM, and includes a reference implementation that supports both scale-up machines and scale-out clusters.

4 The NCNPR Workflow

Computer-aided methods have become central to biomedical research by enabling virtual experiments—most notably molecular docking—which predict binding conformations and affinities between ligands and targets [6, 30, 33]. Docking is widely used for virtual screening of large phytochemical libraries and for drug development, but interpretation requires biochemical context (e.g., GPCRs can adopt multiple functional states and support both orthosteric and allosteric modulation) [26, 49, 51].

IDS reduces the compute burden of large-scale docking by providing scalable access to HPC resources and by prefetching and organizing large ligand and receptor datasets. For the NCNPR use case we integrate curated, multi-modal sources—protein sequences [16], chemical properties [12], and protein 3D structures [3]—into a knowledge graph containing >100 billion facts. This enables much faster virtual screening workflows by narrowing candidate sets before expensive simulations.

The drug re-purposing workflow used in our experiments proceeds as follows:

(1) find proteins related to target protein of interest (Uniprot protein with identifier P29274),
(2) retrieve sequence and structural data for P29274,

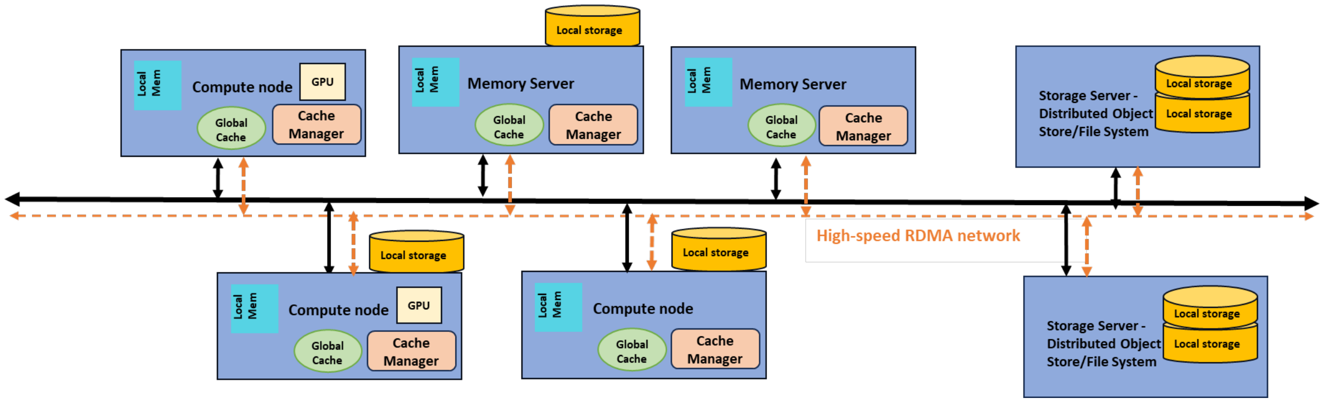


Figure 2: Overview of the global shared client-side cache

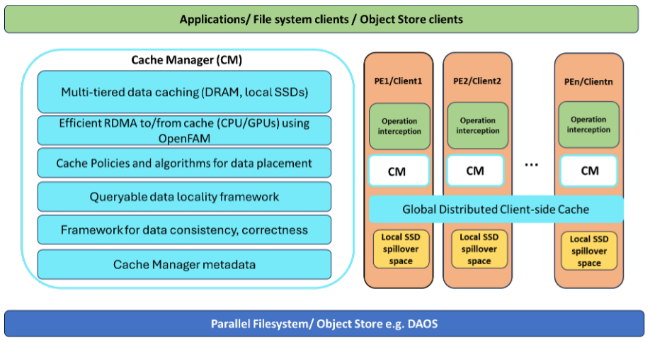

Applications/ File system clients / Object Store clients
Cache Manager (CM)
Multi-tiered data caching (DRAM, local SSDs)
Efficient RDMA to/from cache (CPU/GPUs) using OpenFAM
Cache Policies and algorithms for data placement
Queryable data locality framework
Framework for data consistency, correctness
Cache Manager metadata
PE1/Client1
PE2/Client2
PEn/Clientn
Operation interception
CM
Global Distributed Client-side Cache
Local SSD spillover space
Parallel Filesystem/ Object Store e.g. DAOS

Figure 3: Components of the global cache

(3) assemble candidate compounds that inhibit related proteins,
(4) apply AI to predict drug–target binding affinity (DTBA) and filter candidates, and
(5) perform molecular docking between the filtered compounds and P29274.

Steps 1–4 intentionally prune the search space earlier because they are far cheaper than performing docking. For example, sequence similarity (Smith–Waterman) runs in under a millisecond per comparison [50], DTBA predictions take tenths of a second, whereas docking can take tens of seconds per ligand [8]. Because researchers iterate and refine queries, many successive candidate sets overlap; this reuse motivates caching of docking results.

We integrated IDS with a global distributed cache to address docking latency. By stashing docking outputs, IDS avoids repeated long-running simulations across repeated or subset queries, removing the bottleneck imposed by the slowest rank and substantially reducing wall-clock time for interactive exploration [8].

5 Experimental Setup, Design, and Evaluation

This section describes our experimental setup and metrics. We ran scaling experiments on an HPE Cray EX system (up to 1000 nodes) over Slingshot; compute nodes are dual-socket AMD EPYC Rome (64 cores/socket) with 512 GB RAM and a Clusterstor Lustre backend (12 OSTs, 2 PB). IDS was packaged in an Ubuntu 22.04 Apptainer container including OpenMPI 5 [5], Libfabric 2.1 [2], and libcxi shs-12.0 [4]. We measured end-to-end query latency and FILTER operation time. Scaling runs used 64, 128, and 256 nodes with 32 ranks per node (2048, 4096, 8192 total ranks).

A second, 52-node testbed (compute+memory nodes) connected by Slingshot (25 GB/s) hosted the global cache prototype and smaller scale docking experiments; nodes use dual-socket AMD EPYC 7763 (64 cores/socket), compute nodes have 1 TB RAM, memory nodes 4 TB, and the system runs SLES 15 SP4 with OpenFAM 3.1. This cluster was used to evaluate caching/stashing effects on docking latency.

5.1 Experimental Protocol

To evaluate efficiency and scalability under realistic conditions, we ran a set of experiments described below.

Our NCNPR workflow builds on earlier CGE-based drug discovery work from the COVID-19 effort [38], which integrated multiple public sources into a life-sciences knowledge graph with >100 billion facts. The NCNPR use case augments that graph with protein structural data from the Protein Data Bank (PDB) as needed [3]. This workflow extends the prior pipeline by integrating AI models and HPC-style simulations within IDS, enabling much larger-scale re-purposing searches than typical single-node workflows. Practically, we added (i) a TensorFlow-based DTBA UDF using a pre-trained model that consumes a protein sequence and a SMILES string [32], and (ii) Autodock Vina blind-docking for 3D docking energy calculations [20, 44].

Table 1: Knowledge Graph Dataset Characteristics.

Dataset	Raw Size (disk)	Size (triples)	Source
UniProt	12.7 TB	87.6 Billion	[17]
ChEMBL-RDF	81 GB	539 Million	[28]
Bio2RDF	2.4 TB	11.5 Billion	[10]
OrthoDB	275 GB	2.2 Billion	[47]
Biomodels	5.2 GB	28 Million	[15]
Biosamples	112.8 GB	1.1 Billion	[25]
Reactome	3.2 GB	19 Million	[22]

The inner query performs:

- a search of reviewed proteins,
- retrieval of candidate compounds that inhibit those proteins, and
- filtering by Smith–Waterman sequence similarity, pIC50, and DTBA scores. [1]

Four UDFs are used (Smith–Waterman similarity, pIC50, DTBA prediction, docking) and are intentionally ordered by increasing cost and pruning power. In our runs, approximately 66M UniProt sequences are compared to P29274, with Smith–Waterman averaging < 1 ms per comparison [50], pIC50 costing $1e - 5s$, DTBA taking tenths of a second per prediction, and docking taking multiple seconds per ligand [8]. Because early filters drastically reduce candidates and researchers iterate, many queries reuse overlapping candidate sets, motivating the cache to stash docking outputs and avoid repeated long simulations.

5.2 Performance Analysis

Figure 4(a) shows scaling for the NCNPR drug re-purposing query that includes DTBA and docking UDFs. At first glance the query appears not to scale; detailed analysis (Figure 4(b)) attributes this primarily to docking, while FILTER operations scale well thanks to IDS's reordering and re-balancing of intermediate solutions. Scan/join/merge stages scale until roughly 128 nodes and then plateau, likely because ranks exhaust useful work (for example, at 256 nodes IDS can process >1T edges but the graph contains only $\approx 100B$). Docking (green) is the dominant cost.

The inner query returns relatively few compounds for docking (55 in one reported run), while experiments used 2048, 4096 and 8192 ranks. Docking times per compound vary ($\approx$ 31–44 s), and total query times are 86, 72 and 62 s on 64, 128 and 256 nodes respectively; thus docking dominates end-to-end latency. With many idle ranks, re-balancing is not beneficial in this case; expanding the candidate set would utilize idle resources more effectively. Excluding docking, the remaining execution time is $\approx$ 43, 29 and 19 s for 64, 128 and 256 nodes, indicating reasonable scaling for non-docking stages.

Before docking, execution is dominated by the inner FILTER (Smith–Waterman, pIC50, DTBA). Figure 5 reports FILTER times of $\approx$ 27, 18.5 and 7.7 s at 64, 128 and 256 nodes. DTBA predictions show variance (most $\approx$ 1 s, some longer), illustrating why per-image throughput matters and how IDS can re-balance based on UDF performance.

Because docking is the dominant cost, we implemented a proof-of-concept global, distributed cache. The distributed cache holds the Vina outputs (falls back to a disk stash on a cache miss, and as a last resort on a total miss, re-executes the docking simulation). For this experiment the cache used memory servers on two nodes and the IDS instance ran on two compute nodes.

Table 2 reports that non-cached Vina queries remain stable for low candidate counts but grow sharply as candidate counts and repeated queries increase. At the high end we observe up to a 15× improvement with caching; overall reductions range 5–15×. These significant gains result from caching the most expensive artifact (docking outputs); further improvements are expected as we optimize the cache and broaden the artifacts that are cached.

[1] pIC50 is a widely used pharmacological measure of compound potency, providing a standardized and intuitive way to compare and interpret inhibition data in drug discovery and biochemical research.

Table 2: Query times for various Smith-Waterman thresholds.

Selectivity	Compounds	query time (s) (w/out caching)	query time (s) (with caching)
0.99	56	47.49	8.99
0.90	56	47.66	8.5
0.80	57	47.87	10.51
0.70	57	47.86	9.06
0.60	57	48.08	8.3
0.50	57	51.7	9.23
0.40	121	358.76	28.93
0.20	1129	3847.07	242.85

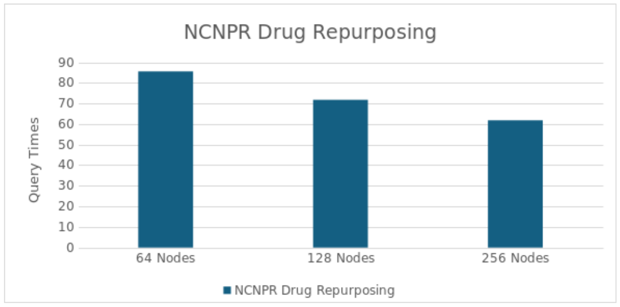
6 Related Work

Our goal is to enable scientists to interact with very large and complex datasets using sophisticated domain-specific computational processes running on HPC systems. The underlying problem is how to enable the scientists' code to achieve data management and movement at scale. To this end, we are informed by prior efforts that share this goal and address it by providing frameworks and programming interfaces for managing data on HPC systems. Our approach allows Python programmers to use arbitrary unmodified Python libraries to perform explorations that leverage IDS to create dynamic complex query workflows consisting of traditional graph searches, pretrained AI models and HPC style simulations. They can further accelerate these workflows by stashing and reusing results [8].

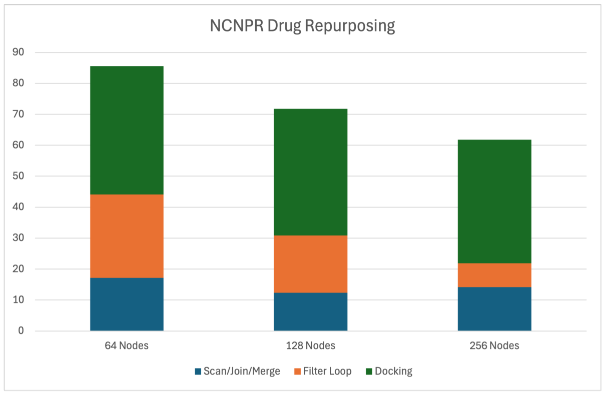
Tang et al. propose Proactive Data Containers (PDC), which is an object-centric data management system that provides an object-centric data model and programming interface for managing data objects on HPC systems [43]. Instead of file-oriented approaches, PDC provides a user-space, distributed service for asynchronous and transparent data transformations and movement across a storage hierarchy. We find PDC particularly promising with regard to its management of proactive data movement across storage devices, and think it offers a potential complement to our current memory-centric caching hierarchy.

Recognizing Python as the de-facto language for writing scientific code, the authors of Data-Centric (DaCe) Python propose a data-centric approach that enables ordinary Python programmers to create high performance programs [52]. DaCe provides an optimization and specialization toolbox for Python users that uses a Stateful DataFlow multiGraphs (SDFG) data-centric intermediate representation to achieve multi-level data movement transformations [11]. The DaCe approach provides programmers with "(restricted)" implementations of popular programming language like Python or MATLAB [11]. We appreciate the transformation-based approach offered by DaCe and SDFG, which we think could be mutually compatible with IDS's sophisticated search capabilities and the transparent caching mechanisms offered by our global cache.

Client-side caching, used in file systems like Lustre and object stores like Redis, is a well-established technique in storage systems [35, 37]. The concept of shared client-side multi-level



(a) NCNPR Drug Re-purposing Query Times



(b) Breakdown of Query Time Components

Figure 4: NCNPR Drug Repurposing Query without our solution: docking performance did not scale.

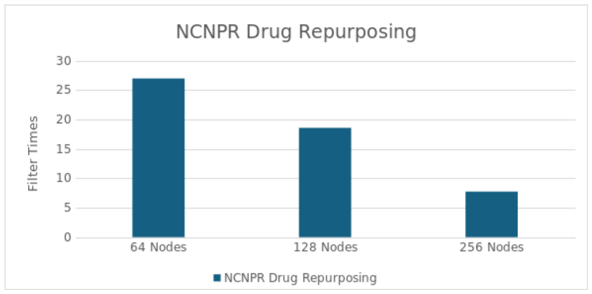


Figure 5: NCNPR Drug Repurposing Filter Times

caches in distributed file systems has been recognized for its ability to enhance scalability by protecting the underlying file system from client activity [14, 18]. Modern High-Performance Computing (HPC) systems now feature complex memory and storage hierarchies with various data movement mechanisms, such as shared burst buffers that leverage client-side resources to handle transient I/O patterns [21, 45]. Cachelib offers a general purpose caching engine for building high performing caches in local node setup [13]. Our cache, however, is distinct in several ways. It is shared, global, and multi-tiered, offering a unified caching layer that can be queried for data locality. This feature enables frameworks to build efficient scheduling algorithms by co-locating computation with data. More importantly, our shared cache is designed to allow direct access to the portion of the cache that resides locally on a node, enhancing performance and reducing data access latency.

With regard to drug discovery, molecular docking is often integrated into workflows that combine a variety of computational methods in order to improve prediction performance [34, 36]. A number of researchers have identified limitations of molecular docking and proposed approaches for addressing them. The value of such workflows and techniques lies in their potential to reduce the time and cost associated with drug discovery [34], and screening large numbers of compounds in a reasonable time is a recognized challenge [36]. However, much prior work focuses on improving approximate scoring functions or improving post-docking processing strategies so as to better identify false-positive and false-negative hits [34, 36]. In contrast, we offer a more general two-fold approach, offering users a general framework that empowers ordinary scientists to apply the resources of a super-computing cluster to query massive scientific data as well as a shared hierarchical cache that scientists can use to stash results to speed time to future results.

7 Summary and Conclusions

In this paper, we discussed the Intelligent Data Search (IDS) framework and how it integrates experimental results into multi-modal knowledge graphs and enables dynamic workflows of computationally complex queries. IDS leverages popular machine learning libraries and the Cray Graph Engine (CGE) to facilitate large-scale exploration and experimentation. Some of the key IDS features include (i) hosting massive multi-terabyte (or petabyte) scale knowledge graphs integrated from multiple sources, (ii) acting as a computational engine capable of sophisticated analytics, (iii) enabling scientists to integrate experimental results into dynamic workflows, and (iv) supporting sophisticated user-defined functions and AI-powered UDFs.

We also discussed how a globally distributed cache can help accelerate computations by caching simulation artifacts both from the workflow and IDS itself. IDS also supports interactive queries and semantic traversal capabilities for data-driven discovery, accelerates domain-specific functions, and executes complex workflows efficiently.

We then demonstrated a proof-of-concept molecular docking workflow, showcasing how IDS can enhance drug discovery processes by integrating AI models and HPC simulations. This workflow, developed in collaboration with the National Center for Natural Products Research (NCNPR), demonstrates IDS's ability to scale and reduce computational latency, significantly improving the efficiency of drug re-purposing studies.

8 Opportunities and Next Steps

While the preceding summary highlights the key findings and their implications, these results also open the door to new avenues for exploration. The following section outlines potential opportunities and proposes concrete next steps to build upon the foundation established in this work.

The first and most logical extension of this work would be to cache more artifacts in the critical path of the molecular docking workflow to minimize latency further. In addition to this, we are in the process of exploring how to cache IDS internal artifacts themselves. These internal artifacts are currently cached in CGE, which, to a large amount, constrains the type of objects that can be cached in it. Significant latency is incurred due to the serialization required to stash objects there. As heterogeneous workflows utilize the framework, the serialization overhead to cache in CGE becomes quite significant. The global cache, enabled by OpenFAM, has no such restrictions and can cache any data.

The next exciting avenue which we believe is worth looking into is exploiting data locality and affinity to enhance the performance of the system. As discussed previously in Section 5.2, as the candidate compounds grow in number, the docking process becomes extremely constrained. With our cache's ability to answer questions about data locality, custom scheduling algorithms can be developed that place IDS's MPI ranks on compute nodes closer to the data they require. We hypothesize that this would result in significant savings in terms of communication and data transfer latencies.

Finally, we envision that IDS, in conjunction with the cache, will enable researchers to potentially share cached results across clusters, running multiple instances of IDS. For example, researchers working on cluster A might run simulations that result in a collection of artifacts that are cached. Other researchers, working on cluster B on a different IDS instance could then leverage to reproduce results, continue investigations etc.

References

[1] [n. d.]. Hugging Face. https://huggingface.co
[2] [n. d.]. libfabric. https://github.com/ofiwg/libfabric
[3] [n. d.]. RCSB Protein Data Bank. https://www.rcsb.org
[4] [n. d.]. shs-libcxi. https://github.com/HewlettPackard/shs-libcxi
[5] 2021. Open MPI: Open Source High Performance Computing. https://www.open-mpi.org
[6] SC Annam, WM Neal, P Pandey, B Avula, K Katragunta, I Husain, SI Khan, I Koturbash, BJ Gurley, IA Khan, and AG Chittiboyina. 2024. A Combined Approach for Rapid Dereplication of Herb–Drug Interaction Causative Agents in Botanical Extracts – A Molecular Networking Strategy To Identify Potential Pregnane X Receptor (PXR) Modulators in Yohimbe. *ACS Omega: American Chemical Society* 9, 52 (Dec 2024), 51394–51407.
[7] Anonymous. 2025. (anonymized due to double-blind review).
[8] Anonymous. 2025. (anonymized due to double-blind review).
[9] Syed Ismail Faizan Barmawer, Gautham Bhat Kumbla, Mashood Abdulla Kodavanji, Clarete Riana Crasta, Sharad Singhal, and Ramya Ahobala Rao. 2024. Client allocation of memory across memory servers. US Patent App. 17/815,366.
[10] Francois Belleau, Marc-Alexandre Nolin, Nicole Tourigny, Philippe Rigault, and Jean Morissette. 2008. Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics* 41 (2008), 706–716. Issue 5. http://www.sciencedirect.com/science/article/pii/S1532046408000415
[11] Tal Ben-Nun, Johannes de Fine Licht, Alexandros N. Ziogas, Timo Schneider, and Torsten Hoefler. 2019. Stateful dataflow multigraphs: a data-centric model for performance portability on heterogeneous architectures. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, Colorado) *(SC '19)*. Association for Computing Machinery, New York, NY, USA, Article 81, 14 pages. doi:10.1145/3295500.3356173
[12] A Patrícia Bento, Anna Gaulton, Anne Hersey, Louisa J Bellis, Jon Chambers, Mark Davies, Felix A Krüger, Yvonne Light, Lora Mak, Shaun McGlinchey, Michal Nowotka, George Papadatos, Rita Santos, and John P Overington. 2014. The ChEMBL bioactivity database: an update. *Nucleic acids research* 42, Database issue (January 2014), D1083—90. doi:10.1093/nar/gkt1031
[13] Benjamin Berg, Daniel S. Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, and Gregory R. Ganger. 2020. The CacheLib Caching Engine: Design and Experiences at Scale. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.
[14] M. Blaze and R. Alonso. 1992. Dynamic hierarchical caching in large-scale distributed file systems. In *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*. 521–528. doi:10.1109/ICDCS.1992.235001
[15] Vijayalakshmi Chelliah, Nick Juty, Ishan Ajmera, Raza Ali, Marine Dumousseau, Mihai Glont, Michael Hucka, Gaël Jalowicki, Sarah Keating, Vincent Knight-Schrijver, Audald Lloret-Villas, Kedar Nath Natarajan, Jean-Baptiste Pettit, Nicolas Rodriguez, Michael Schubert, Sarala M. Wimalaratne, Yangyang Zhao, Henning Hermjakob, Nicolas Le Novère, and Camille Laibe. 2015. BioModels: ten-year anniversary. *Nucl. Acids Res.* 43 (2015), D542–D548. Issue D1. doi:10.1093/nar/gku1181
[16] Uniprot Consortium. 2017. UniProt: the universal protein knowledgebase. *Nucleic Acids Research* 45, D1 (2017), D158. doi:10.1093/nar/gkw1099 arXiv:https://academic.oup.com/nar/article-pdf/45/D1/D360/22453373/gkw1099.pdf
[17] Uniprot Consortium. 2019. UniProt: the universal protein knowledgebase. *Nucleic Acids Research* 47 (2019), D506.
[18] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. 1994. Cooperative caching: using remote client memory to improve file system performance. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation (OSDI '94)*. USENIX Association, USA, 19–es. event-place: Monterey, California.
[19] daos stack. 2020. DAOS Storage Stack. https://github.com/daos-stack/daos Accessed: 2020-08-27.
[20] J. Eberhardt, D. Santos-Martins, A. F. Tillack, and S. Forli. 2021. AutoDock Vina 1.2.0: New Docking Methods, Expanded Force Field, and Python Bindings. *Journal of Chemical Information and Modeling* (2021).
[21] Dave Henseler et al. 2016. Architecture and Design of Cray DataWarp. In *Proceedings of the Cray User Group (CUG) Conference 2016*. https://cug.org/proceedings/cug2016_proceedings/includes/files/pap105s2-file1.pdf Accessed: 2025-04-08.
[22] Antonio Fabregat, Steven Jupe, Lisa Matthews, Konstantinos Sidiropoulos, Marc Gillespie, Phani Garapati, Robin Haw, Bijay Jassal, Florian Korninger, Bruce May, Marija Milacic, Corina Duenas Roca, Karen Rothfels, Cristoffer Sevilla, Veronica Shamovsky, Solomon Shorser, Thawfeek Varusai, Guilherme Viteri, Joel Weiser, Guanming Wu, Lincoln Stein, Henning Hermjakob, and Peter D'Eustachio. 2018. The Reactome Pathway Knowledgebase. *Nucleic Acids Res* 46, D1 (Jan 2018), D649–D655. doi:10.1093/nar/gkx1132
[23] K. Harms. 2023. DAOS is your Future Distributed Asynchronous Object Store. Presented at the Argonne Leadership Computing Forum Hands-on HPC Workshop. https://www.alcf.anl.gov/sites/default/files/2023-10/DAOS-ALCF-HoW23.pdf Accessed: 2024-09-30.
[24] M. Hennecke. 2023. Understanding DAOS Storage Performance Scalability. In *Proceedings of the HPC Asia 2023 Workshops* (Raffles Blvd Singapore). ACM, 1–14. doi:10.1145/3581576.3581577
[25] Simon Jupp, James Malone, Jerven Bolleman, Marco Brandizi, Mark Davies, Leyla Garcia, Anna Gaulton, Sebastien Gehant, Camille Laibe, Nicole Redaschi, Sarala M Wimalaratne, Maria Martin, Nicolas Le Novère, Helen Parkinson, Ewan Birney, and Andrew M Jenkinson. 2014. The EBI RDF platform: linked open data for the life sciences. *Bioinformatics* 30, 9 (05 2014), 1338–1339. doi:10.1093/bioinformatics/btt765
[26] Vsevolod Katritch, Vadim Cherezov, and Raymond C. Stevens. 2013. Structure-function of the G protein–coupled receptor superfamily. *Annual Review of Pharmacology and Toxicology* 53, 1 (Jan 2013), 531–556. doi:10.1146/annurev-pharmtox-032112-135923
[27] Kristyn Maschhoff, Robert Vesse, and James Maltby. 2015. Porting the Urika-GD Graph Analytic Database to the XC30/40 Platform. In *Cray User Group Conference (CUG '15)*. Chicago, IL.
[28] David Mendez, Anna Gaulton, A Patrícia Bento, Jon Chambers, Marleen De Veij, Eloy Félix, María Paula Magariños, Juan F Mosquera, Prudence Mutowo, Michał Nowotka, María Gordillo-Marañón, Fiona Hunter, Laura Junco, Grace Mugumbate, Milagros Rodriguez-Lopez, Francis Atkinson, Nicolas Bosc, ChrisJ Radoux, Aldo Segura-Cabrera, Anne Hersey, and Andrew R Leach. 2018. ChEMBL: towards direct deposition of bioassay data. *Nucleic Acids Research* 47, D1 (11 2018), D930–D940. doi:10.1093/nar/gky1075 arXiv:https://academic.oup.com/nar/article-pdf/47/D1/D930/27437436/gky1075.pdf
[29] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 561–577.

[30] Garrett M. Morris and Michael Lim-Wilby. 2008. Molecular Docking. In *Molecular Modeling of Proteins*, Andreas Kukol (Ed.). Methods in Molecular Biology, Vol. 443. Humana Press, New York, NY, 365–382. doi:10.1007/978-1-59745-177-2_19

[31] NVIDIA. 2025. RDMA Architecture Overview. https://docs.nvidia.com/networking/display/rdmaawareprogrammingv17/rdma+architecture+overview Accessed: 2025-04-08.

[32] Hakime Öztürk, Arzucan Özgür, and Elif Ozkirimli. 2018. DeepDTA: deep drug–target binding affinity prediction. *Bioinformatics* 34, 17 (2018), i821–i829.

[33] P. Pandey, K. Roy, and R. Doerksen. 2015. Protein structure-based virtual screening led to identification of novel natural product-derived hits as cannabinoid receptor 1 modulators. In *Abstracts of Papers of the American Chemical Society*, Vol. 249. American Chemical Society, 1155 16th St, NW, Washington, DC 20036, USA.

[34] Luca Pinzi and Giulio Rastelli. 2019. Molecular Docking: Shifting Paradigms in Drug Discovery. *International Journal of Molecular Sciences* 20, 18 (Jan. 2019), 4331. doi:10.3390/ijms20184331 Number: 18 Publisher: Multidisciplinary Digital Publishing Institute.

[35] Yingjin Qian, Xi Li, Shuichi Ihara, Andreas Dilger, Carlos Thomaz, Shilong Wang, Wen Cheng, Chunyan Li, Lingfang Zeng, Fang Wang, Dan Feng, Tim Süß, and André Brinkmann. 2019. LPCC: Hierarchical Persistent Client Caching for Lustre. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19)*.

[36] Giulio Rastelli and Luca Pinzi. 2019. Refinement and Rescoring of Virtual Screening Results. *Frontiers in Chemistry* Volume 7 - 2019 (2019). doi:10.3389/fchem.2019.00498

[37] Redis. 2011. *Redis: A Distributed Key-Value Store*. Redis Inc. https://redis.io/

[38] Christopher Rickett, Kristyn Maschhoff, and Sreenivas Sukumar. 2020. Massively Parallel Processing Database for Sequence and Graph Data Structures Applied to Rapid-Response Drug Repurposing. In *IEEE BigData 20202*.

[39] Christopher Rickett, Kristyn Maschhoff, and Sreenivas Sukumar. 2021. Optimizing the Cray Graph Engine for Performant Analytics on Cluster, SuperDome Flex, Shasta Systems and Cloud Deployment. In *Cray User Group Conference (CUG '21)*.

[40] S. Singhal et al. 2022. OpenFAM: A Library for Programming Disaggregated Memory. In *OpenSHMEM and Related Technologies: OpenSHMEM in the Era of Extreme Heterogeneity*. Lecture Notes in Computer Science, Vol. 13694. Springer, 18–32. doi:10.1007/978-3-031-22524-6_2

[41] S. Singhal et al. 2023. OpenFAM: Programming Disaggregated Memory. *Concurrency and Computation: Practice and Experience* 35, 5 (2023), e7291. doi:10.1002/cpe.7291

[42] Foteini Strati, Xianzhe Ma, and Ana Klimovic. 2024. Orion: Interference-Aware, Fine-Grained GPU Sharing for ML Applications. In *Proceedings of the Nineteenth European Conference on Computer Systems*. 1075–1092.

[43] Houjun Tang, Suren Byna, François Tessier, Teng Wang, Bin Dong, Jingqing Mu, Quincey Koziol, Jerome Soumagne, Venkatram Vishwanath, Jialin Liu, and Richard Warren. 2018. Toward Scalable and Asynchronous Object-Centric Data Management for HPC. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 113–122. doi:10.1109/CCGRID.2018.00026

[44] O. Trott and A. J. Olson. 2010. AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization and multithreading. *Journal of Computational Chemistry 31* (2010).

[45] Teng Wang, Kathryn Mohror, Adam Moody, Kento Sato, and Weikuan Yu. 2016. An Ephemeral Burst-Buffer File System for Scientific Applications. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 807–818. doi:10.1109/SC.2016.68 ISSN: 2167-4337.

[46] Weka. 2023. Weka Architecture White Paper. https://www.weka.io/wp-content/uploads/resources/2023/03/weka-architecture-white-paper.pdf Accessed: 2025-04-08.

[47] Evgeny M Zdobnov, Fredrik Tegenfeldt, Dmitry Kuznetsov, Robert M Waterhouse, Felipe A Simão, Panagiotis Ioannidis, Mathieu Seppey, Alexis Loetscher, and Evgenia V Kriventseva. 2019. OrthoDB v10: sampling the diversity of animal, plant, fungal, protist, bacterial and viral genomes for evolutionary and functional annotations of orthologs. *Nucleic Acids Research* 47, D1 (2019), D807–D811. doi:10.1093/nar/gky1053

[48] Bo Zhang, Philip E. Davis, Nicolas Morales, Zhao Zhang, Keita Teranishi, and Manish Parashar. 2023. Optimizing Data Movement for GPU-Based In-Situ Workflow Using GPUDirect RDMA. In *Euro-Par 2023: Parallel Processing: 29th International Conference on Parallel and Distributed Computing* (Limassol, Cyprus). Springer-Verlag, Berlin, Heidelberg, 323–338. doi:10.1007/978-3-031-39698-4_22

[49] M. Zhang, T. Chen, X. Lu, X. Lan, Z. Chen, and S. Lu. 2024. G protein-coupled receptors (GPCRs): advances in structures, mechanisms, and drug discovery. *Signal Transduction and Targeted Therapy* 9, 1 (Apr 2024), 88. doi:10.1038/s41392-024-01803-6

[50] Mengyao Zhao, Wan-Ping Lee, Erik P Garrison, and Gabor T Marth. 2013. SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications. *PLoS One* 8, 12 (2013). doi:10.1371/journal.pone.0082138

[51] C. Zhu, X. Lan, Z. Wei, J. Yu, and J. Zhang. 2024. Allosteric modulation of G protein-coupled receptors as a novel therapeutic strategy in neuropathic pain. *Acta Pharmaceutica Sinica B* 14, 1 (Jan 2024), 67–86. doi:10.1016/j.apsb.2023.11.003

[52] Alexandros Nikolaos Ziogas, Timo Schneider, Tal Ben-Nun, Alexandru Calotoiu, Tiziano De Matteis, Johannes de Fine Licht, Luca Lavarini, and Torsten Hoefler. 2021. Productivity, Portability, Performance: Data-Centric Python. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.