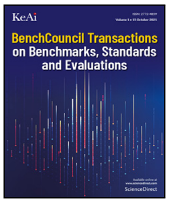
Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/



# SAIBench: Benchmarking AI for Science

Yatao Li [a,b,c,*], Jianfeng Zhan [a,b]

[a] *Institute of Computing Technology Chinese Academy of Science, No. 6 Kexueyuan South Road, Haidian District, 100190, Beijing, China*
[b] *University of Chinese Academy of Sciences, No. 19 (A) Yuquan Road, Shijingshan District, 100049, Beijing, China*
[c] *Microsoft Research Asia, Building 2, No. 5 Dan Ling Street, Haidian District, 100080, Beijing, China*

ARTICLE INFO

*Keywords:*
Scientific computing
Artificial intelligence
Benchmarking

ABSTRACT

Scientific research communities are embracing AI-based solutions to target tractable scientific tasks and improve research work flows. However, the development and evaluation of such solutions are scattered across multiple disciplines. We formalize the problem of scientific AI benchmarking, and propose a system called SAIBench in the hope of unifying the efforts and enabling low-friction on-boarding of new disciplines. The system approaches this goal with *SAIL*, a domain-specific language to decouple research problems, AI models, ranking criteria, and software/hardware configuration into reusable modules. We show that this approach is flexible and can adapt to problems, AI models, and evaluation methods defined in different perspectives. The project homepage is https://www.computercouncil.org/SAIBench.

## 1. Introduction

Artificial Intelligence has seen continuous and significant advancements over the past years, with Deep Learning methods being arguably the most representative and focused on. Blessed by the ever-increasing computation power in AI accelerators and general-purpose architectures alike, new AI paradigms and models are proposed which greatly improve the scalability, flexibility, and applicability of this data-driven approach. As a result, the IT industry is welcoming AI-powered solutions, integrating them into existing data processing pipelines that will otherwise require human intervention or prohibitive computation cost. This trend is also propagating into scientific research communities, as researchers are gaining interest in leveraging state-of-the-art AI solutions to tackle equally if not more difficult tasks, hence AI for Science.

From a bird's eye view, a scientific research activity can be mechanical or creative. A mechanical research activity can be algorithmically specified, with quantized or computationally verifiable input/output. On the other hand, a creative research activity breaks out of a mechanical system, for example, by defining a new problem or introducing ideas hard to quantify. In this work, we call a computationally verifiable research task a "tractable scientific task". That said, an AI for Science solution is introduced to bring improvements into the scientific research workflow and is usually targeting towards a tractable scientific task, such as:

- Mathematical Problem Solving — to solve mathematically well-defined problems.
- Pattern Matching — to classify, identify patterns, and detect region-of-interest in high volume scientific data.
- Prediction — to compute future world states, given an initial snapshot of the world state and evolving rules.
- Artifact enhancement — to improve the quality of data acquired from imperfect observations, e.g. incomplete, fragmented, noisy sensor data.
- Control — to use actuators to drive sensor readings into desired states, despite the imperfection of both.
- Hypothesis and Confirmation — to propose a theory (e.g. equations) that conforms with the observations.

Examples of these tasks are shown in Table 1. The term "AI for Science" is also conventionally deemed as an ensemble of vertical fields and tasks [1]. However, we argue that to fully realize the potential of AI for Science, it is not enough to cherry-pick an AI method, match it against a specific task, and heuristically compare it with existing methods. One strength of AI methods is that they abstract away the problem details and mathematical procedures, into generic functions that transform inputs into outputs — that is, every AI model possesses the potential to adapt to other tasks, some (for example, neural networks) even being universal approximators. Science is vast, and AI methods are many. A single effort to evaluate a taskmethod pair would leave other research communities unaware, of both the potential tasks that a model is capable of processing and potential models that can be applied to a task. This problem is exaggerated by the fact that the AI research is moving forward fast, that by the time a specific method is

* Corresponding author at: Institute of Computing Technology Chinese Academy of Science, No. 6 Kexueyuan South Road, Haidian District, 100190, Beijing, China.
*E-mail addresses:* yatli@microsoft.com (Y. Li), jianfengzhan@ict.ac.cn (J. Zhan).

https://doi.org/10.1016/j.tbench.2022.100063
Received 15 April 2022; Received in revised form 11 May 2022; Accepted 11 May 2022
Available online 24 May 2022

**Table 1**
Examples of tractable scientific tasks.

| | |
|---|---|
| Mathematical Problem Solving | Partial derivative equations |
| | General matrix multiplication |
| | Matrix decomposition |
| | Integration |
| | Monte Carlo methods |
| Pattern Matching | Species Classification |
| | Event Identification [2] |
| | Climate Analysis [3] |
| | Anomaly Detection |
| Prediction | High-Energy Particle Simulation |
| | Molecular Dynamics |
| | Fluid Dynamics |
| | Protein Folding |
| Artifact Enhancement | Genome Sequence Alignment |
| | Astronomy Image Enhancement |
| | Medical Image Enhancement |
| | MRI Reconstruction |
| Control | Tokamak Plasma Control [4] |
| | Sensor Triggering |
| Hypothesis and Confirmation | Automatic Physics Laws Discovery |
| | Symbolic Regression |

picked up by a scientific computing task, or a task is adapted to an AI method, it may be already bested by then state-of-the-art.

To help the scientific research communities as a whole systematically absorb and integrate the advancements of AI research, and to avoid repeated efforts in development and evaluation, we propose *SAIBench*, a system that bridges scientific computing tasks and AI methods, and automatically benchmarks every sensible combination, collects performance metrics, and projects it back into rankings proper to each research community. Research groups of different backgrounds can focus on their needs while taking advantage of other benchmarking building blocks, without having to re-invent end-to-end evaluation processes.

The rest of this article is organized as follows. We first define the problem of scientific AI benchmarking in Section 2. In Section 3 we discuss the methodology, goal, and challenges. Section 4 elaborates our system design, including the details of each component. We showcase end-to-end scenarios involving multiple modules in Section 5.

## 2. Problem definition

Here we define the problem of scientific AI benchmarking. To begin with, we have a set of tractable scientific tasks as defined in the previous section, and an array of AI methods, each needs to be trained to solve a specific problem. To evaluate a method for such tasks, different scientific communities have different criteria. For example, instruments in High Energy Physics generate zettabytes of data, and the training data for AI models is virtually unlimited. An AI method could thus focus on throughput, time-to-solution, sample selection, etc. Meanwhile, for Biology and Life Sciences, sometimes there are just a few hundred data points, requiring high sample efficiency, and a strong ability to generalize and extrapolate onto unseen problem configurations.

Nonetheless, the qualification of a method can be categorized as follows:

- **Defined by Problem Class.** For purely computational tasks such as mathematical problem solving, it is preferable to target against classes of problems to see how the method performs under each set of mathematical constraints. For example in equation solving, it is desired to study how a method behaves for both stiff and non-stiff systems, where both types contain their problem class definitions.
- **Defined by Problem Setting.** Compared to purely mathematical problem classes, this type of problem definition usually embodies specific constraints under a class to match a physical setting. Scientific research communities have established well-respected problem settings that have been practiced and confirmed. This allows computational methods to interoperate with real-world experiments, as specific experimental settings can be virtually replicated.
- **Defined by Problem Cases.** For some tasks we are only interested in a narrow range within the whole problem space. Most data-driven tasks fall into this category, where the typical workloads of a task are defined by collected and/or labeled data. There are also "golden standards" defined in research fields, which are computational methods with superior accuracy and other desirable properties, at expensive computational costs. These methods are then used to collect data for very specific problem cases so that other faster but less accurate methods can be developed and evaluated.

**Table 2**
Examples of qualification criterion.

| | |
|---|---|
| Problem class | Boundary Value Problem |
| | Stochastic Differential Equations |
| | Many-body Interactions |
| | Positive Definite Matrix Decomposition |
| Problem setting | Temperature and pressure dependence of alanine dipeptide |
| | Straight wire Magnetostatics |
| | Community Atmosphere Model (CAM5) [5] Simulation |
| Problem case | ANI-1x [6], GDB-17 [7] |
| | OASIS [8] |

This categorization is not mutually exclusive though, as some tasks require more than one qualification criteria to properly define the problem. For example, a robotic control algorithm can be tested both in a simulated setting and on data points collected from real-world sensors. Nevertheless, the principle is that this categorization describes the hierarchy of problem definition — the more the definition leans towards the former (problem classes), the more computation is required; On the other hand, the more towards the latter (problem cases), the more data. Furthermore, the problem definition serves as a specification for the AI model behavior, for both training and testing.

Examples of these qualifications are shown in Table 2. However, AI-based methods likely require training, so the problem definition of all three types must be reduced to case-by-case training data points — for a problem class, the problem definition should generate independent problem instances that sufficiently cover the problem space. For a problem setting, the problem definition should generate state snapshots that conform to the constraints. For data-driven problem cases, the problem definition should simply enumerate from the dataset.

Furthermore, the evaluation of a method depends on the problem definition generating tests. For each test case, the performance is represented with a cost function. For a mathematical problem instance, the cost function can be the error against ground truth solution, or error against equality constraints [9,10]. For simulation settings, the cost function can be obtained by comparing the performance metrics derived from such experiments, as shown by previous works on specific tasks [11–13]. Lastly, for data-driven problem cases, the dataset can be split into training and test sets, and the cost function is the loss function applied to the test set.

Finally, it is crucial to realize that different benchmarking communities use the word "performance" to refer to different concepts. Scientific AI benchmarking concerns not only the accuracy of AI models but also the computation cost. The computation cost can be further broken down into two phases: (1) the cost for a model to reach certain accuracy, and (2) once the model is properly trained, the cost of using the model for inference tasks. For the first phase, the standard practice is to measure training time (wall clock or total CPU/GPU time) against the best/mean/worst accuracies, and for the second, the throughput/latency etc. for completing the inference tasks.

Moreover, for both phases, we can investigate the system performance with standard parallel computing benchmarking techniques [14] to expose different performance characteristics of a solution, for example, time-to-solution or cost-efficiency.

## 3. Methodology

The main goal of *SAIBench* is to build an inclusive and interconnecting environment for all the relevant research efforts, including problem definition, AI method, training algorithm, software and hardware environment, metric definition and ranking definition, and deliver benchmarking result efficiently with given computation resources. The desiderata brought by this goal is multifold.

We need to design the system with a modular paradigm and provide friendly programming interfaces for different modules. It should handle the impedance mismatches between different programming languages and environments while maintaining consistent standards. This is traditionally implemented with language bindings (for example, the computational chemistry package NWChem [15] can either execute its own scripting language, or be controlled by a Python language binding) or file-based inter-process communication, which is suboptimal because different programming environments may have incompatible constructs that cannot be bound into a single process, and distributed computing modules cannot be modeled easily.

A module should be self-descriptive so that the system can automatically discover benchmarking tasks it can participate in, so in addition to modular interfaces targeting benchmarking tasks, there should be a protocol for modules to exchange metadata and relate to each other. It is challenging to design such a protocol because it has to be generic, extensible, and yet carrying concrete meanings. For example, if we model the input/output of an AI model as tensors of required dimensions, it places strict constraints on what the AI model can solve, and the system will not be able to associate this AI model with even slightly incompatible tensors, not to mention non-tensor data that has to be converted to adapt. On the other hand, if we simply attach a textual description to each module, it would be too hard for machine-understanding, and require human in- tervention to develop the connections. To this end, machine-understandable flexibility and extensibility are needed, to enable modules to cooperate less rigidly. The previous example shows how a module for an AI model should describe itself. Similarly, for a problem definition, it should programmatically set up the training and test fixtures, and conduct the experiments. This way all the three types of problem definitions previously can be normalized and become accessible to AI models. In addition, it should expose metadata that allows the system to inspect the execution workflow, and identify tasks that can be completed by other modules. This type of meta-programming is practiced in programming language research and recently machine learning frameworks, implemented in declarative languages and domain-specific languages (DSLs) [16], yet largely unexplored in scientific computing, where most execution engines take a parse–interpret–execute approach [15,17].

As we discussed above, the system is not a single benchmark, but a collection of such, to be projected back to each research field and aggregated by a ranking criterion. Conflict of interests naturally arises, for example, to favor speed vs. to favor accuracy, first principle metrics vs. a particular set of derived properties. The system should be able to allow different perspectives of the same metrics and provide an interface for ranking modules to declare their preferences.

The performance of an end-to-end AI solution to a tractable scientific task depends on multiple aspects, including the AI model, the training algorithm, the computing software stack, the empowering hardware, and so on. These factors do not contribute to the final performance linearly, for example, a particular AI model may have the best work-precision properties under one hardware configuration but not the others. It is thus desirable to consider all these factors as benchmarking hyperparameters. There are several implications brought by this requirement. The AI module implementation should be declarative instead of being bound to a specific software/hardware stack; The software stack module should declare the capabilities (e.g. matrix multiplication and backward propagation) so that the system finds compatible model-software pairs; Also, the software stack module should describe the hardware compatibility and accept a standardized hardware configuration descriptor, so that the system can automatically schedule scalability tests.

With all the components modularized and parameterized, the whole benchmarking workflow can be formulated as follows. Each type of module introduces some *dimensions* to the benchmarking task, and the goal is to enumerate and test against the Cartesian product of all such dimensions, where each point in the problem space represent the combination of a specific task, solver, metrics, software and hardware configuration. This allows the modules to advertise themselves, discover the others, and therefore reuse data and interact with each other, without knowing them beforehand. This paradigm aligns well with the FAIR guiding principles for scientific data management [18], which suggests that scientific data should have findability, accessibility, interoperability, and reusability. This is the key difference between the methodology of this work and previous AI benchmarking and scientific benchmarking systems, where the benchmarked scenarios are pre-determined workload and model combinations, and the addition of a new AI model or dataset would not be automatically discovered and reused by existing modules in the system and has to be scripted by a programmer.

Last but not least, because the system automatically discovers potential benchmarking tasks, it is desired that the system can concurrently schedule computation resources to them. As different benchmarking tasks may require different computing environments, it is crucial that the system can elastically provision the environment for each task in a standardized manner, including the operating system, runtime libraries, setup scripts, and test fixture data. The challenge lies in how to design the system to efficiently support such needs and minimize the deployment overhead.

## 4. System design

In this section, we illustrate the overall design of the system and tap into each system component, and discuss how to address the aforementioned challenges. The architecture of the system is depicted in Fig. 1. The workflow is straightforward. The planner pulls all modules from the module repository and joins them into feasible tuples according to the metadata descriptors. The execution plan is then dispatched to the elastic

computing platform which provides storage, processors, and accelerators, where each benchmarking task tuple is executed in a "benchmarking pod". The purpose of the BenchPod is to provide task-level isolation to computation resources, a communication endpoint to interact with the planner, and experiment orchestration. A problem definition module either generates data on-the-fly or retrieves a well-known dataset into the BenchPod instance. The hardware definition module acquires hardware resources. The software definition module constructs a containerized environment, based on a standardized software package requirement descriptor. The entry point of the container is a shim program provided by the BenchPod instance that orchestrates the actual execution of the solver, metric collection, and aggregation.

### 4.1. SAIL: Scientific AI domain-specific language

Previous AI benchmarking systems either implicitly define a series of built-in modules [19,20], or expose a markup language schema to define modules [21]. For better programmability, discoverability, and user ergonomics, we propose to define modules with an embedded domain-specific language (eDSL) called SAIL. The eDSL is implemented as a Python package so that a module implementer can take advantage

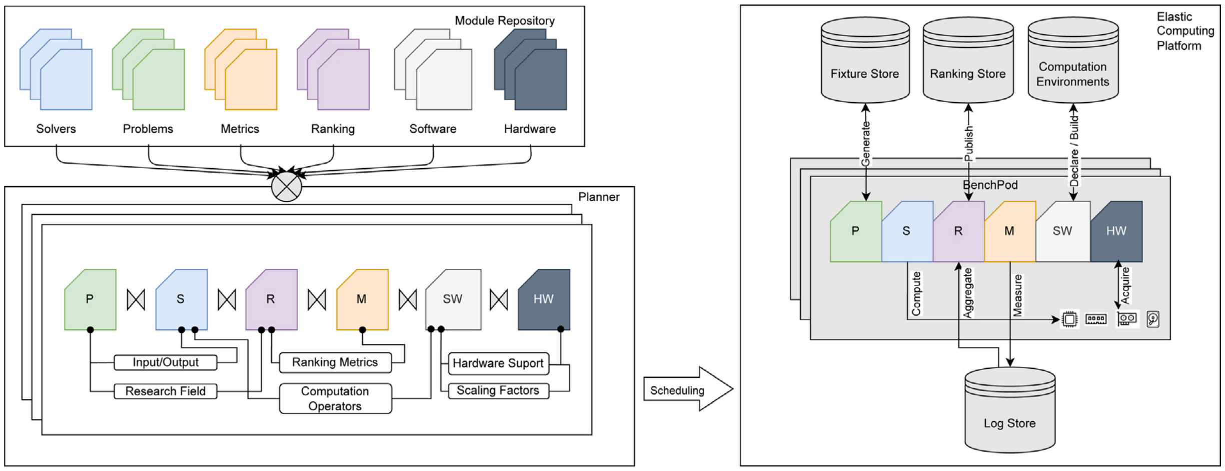


**Fig. 1.** System architecture.

```
@ProblemDefinition
def MNIST(epochs):
  data_type   = Tensor(28, 28)
  label_type  = Tensor(10)
  train_data  = Input("train-images.idx3-ubyte", data_type)
  train_label = Input("train-labels.idx1-ubyte", Scalar).OneHot(label_type)
  test_data   = Input("t10k-images.idx3-ubyte", data_type)
  test_label  = Input("t10k-labels.idx1-ubyte", Scalar).OneHot(label_type)
  for x,y in zip(train_data, train_label):
    Train.Classify(x,y)
  for x,y in zip(test_data, test_label):
    Test.Classify(x,y)
```

**Fig. 2.** MNIST Problem Definition.

of modern IDE features such as auto-completion and type checking while writing the module definition. To design an eDSL means that the desired features must be retrofitted into the target language. To achieve this, we take advantage of various Python language constructs that best fit the required features. Some features can be implemented with static analysis, for example, we use Python decorators to identify module entry points. This way we can easily scan for modules with reflection, and build our module repository. We use Python classes to represent type descriptors for our type system, which is a dual-role construct that both encodes the type information for static analysis, and dispatches code during benchmarking runtime. Benchmarking concepts are modeled as well-known global objects, and the methods attached to them represent benchmarking primitives. This gives a hint to the user that these concepts are stateful, and the primitives can function as both computation routine and data storage. Finally, we use declarative methods to construct the computation graph for AI models. Table 3 shows some language construct examples.

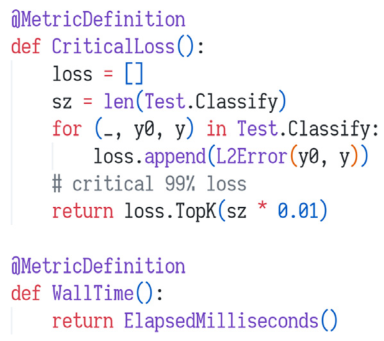
The module script, rather than directly executed in a Python interpreter, is first sent to the SAIL parser. The SAIL parser substitutes the actual execution logic with computation nodes and connects the nodes with computational dependencies to construct the computation graph, similar to the tape-recording technique in automatic differentiation frameworks [22]. The parser then analyzes the computation graph and synthesizes actual benchmarking code. The eDSL provides its own type system with both tensors and symbolic equations as first-class citizens, and helper functions to help connect different modules. In fact, with proper type inference, there is even no need to explicitly declare the input/output types of a module.

The flexibility of a scripting language also simplifies module definitions, for example, Fig. 2 illustrates a "hello world" problem definition module — the MNIST [23] image classification problem. This is a typical "defined by cases" problem as we illustrated in Section 2. The definition of this problem reads from four input files, joins them into pairs, and declares the data points and associated classification tasks into **Train** and **Test** collections. Note that the existence of both train and test collections is not necessary for some kinds of problems — for example, a PDE "problem class" definition may define a few equations in the test collection and expect a solver to accomplish the task without training or hints.

Note that the problem definition of MNIST resembles a machine learning training loop — but not entirely. The key point is that it only defines the problem, and does not try to solve or evaluate the results. This allows us to plug different evaluation metrics into the workflow. For example, the Machine Learning community traditionally focuses on the average performance over the whole dataset, while in a production critical environment, one may prefer to evaluate 99%

**Table 3**
Examples of SAIL language constructs.

| Feature | Construct | Instances |
|---|---|---|
| Module entry points | Decorators | @ProblemDefinition<br>@MetricDefiniton . . . |
| Type descriptors | Classes | **class** Tensor<br>**class** Scalar . . . |
| Concepts and Primitives | Well-known<br>Global Objects | **Train**. Classify<br>**Model**. Predict<br>**Test**. Compare . . . |
| AI models | Declarative methods | Pipeline<br>Linear<br>Relu<br>Softmax . . . |



```python
@MetricDefinition
def CriticalLoss():
    loss = []
    sz = len(Test.Classify)
    for (_, y0, y) in Test.Classify:
        loss.append(L2Error(y0, y))
    # critical 99% loss
    return loss.TopK(sz * 0.01)

@MetricDefinition
def WallTime():
    return ElapsedMilliseconds()
```

**Fig. 3.** Custom Evaluation Metric Definition.

percentile precision, or a hard fail condition, as shown in Fig. 3. Also shown in the code is a simple timer metric, and a task can be evaluated with multiple metrics. For example, the two in the code will combine into a work–critical loss 2D graph. For iterative tasks, a metric will also be evaluated iteratively, and a module can choose to keep states across multiple iterations, memorize the data points or obtain the average, etc.

Even for the same task, different research communities have different interests in performance evaluation. For example, scientific research groups focus on the quality of the end result, while computer system researchers focus on system performance metrics, such as throughput and latency [24]. This is why we further split the ranking module from problems and metrics. A ranking module can reference multiple metrics and aggregate them to obtain a total order, or implement a comparison between two instances to obtain partial order.

Another advantage of this approach is that the module definition can take input parameters and programmatically generate configurations. For example, in Fig. 4 we define how to pick the correct docker image tag for TensorFlow based on the hardware configuration, which is hard to model with a markup language. This also allows us to define generic AI modules that adapt to different input sizes and types and suggest hyper-parameter values. Fig. 5 shows the definition of a simple neural network, which not only defines the computation graph, but also the intended tasks, input/output type conversion, and layer width suggestions, so that the planner can grid search this hyperparameter. Also shown in the code snippet are two type converters, when combined, can automatically convert the input of an atom sequence into a single concatenated tensor.

## 4.2. Automatic benchmarking task discovery

As discussed before, the module definitions are not used for the actual execution of the benchmarking tasks. Rather, they are metaprogramming constructs that can be seen as a "dryrun" for the actual benchmarking. The system scans all python files and uses reflection to identify module entry points, and create records for them in the module repository. The system then enumerates all the modules from the repository and constructs candidate test fixtures, which are tuples of different kinds of modules. For each candidate tuple, the system executes the modules in it, providing input parameters, and extracting information such as the problems a model can solve, the research field of a problem, the suggested hyperparameters, and compatible metrics for a kind of task and so on. The execution order is determined by the type of modules and the implied dependencies — problem definitions execute first because they generally do not depend on other modules, and populate the metadata required to associate metrics and ranking. During execution, the system maintains the context for the current test fixture and accumulates the metadata from already executed modules in the candidate tuple, and later modules can either be filtered by metadata matching (for example, by matching data types) or actively reject the context. This is demonstrated in earlier examples, where a module can use the DSL primitive **Fail** to indicate that it does not know how to solve the problem, or the hardware does not support the current software configuration. Additionally, the system builds a graph where the nodes are data types and edges are converters, and employs breadth-first search to also allow type converter composition so that multiple converters can work together to relax type constraints and improve module compatibility. This process is akin to the inner join operation in relational databases, and the system builds complete tuples of the modules as test scenarios. Apart from automatic discovery, a module can also explicitly declare relationships with other modules so as to narrow down the search space. The logic is presented in algorithm 1.

**Algorithm 1:** Benchmarking Task Discovery.

```
Input: eDSL source files [src]
Output: Test scenarios [test]
repo ← ϕ;
// 1. Module discovery
foreach s : src do
    ast ← parse(s);
    foreach m : methods(ast) do
        if decorated(m, ProblemDefinition) then
            repo[0].append(m);
        if decorated(m, MetricDefinition) then
            repo[1].append(m);
        // Scan for other modules...
ctx ← ϕ; test ← ϕ; iters ← iterators(repo);
i ← 0;
// 2. Task discovery
while i ≥ 0 do
    if i = N then
        test.append(ctx);
        ctx.pop();
        i ← i − 1;
    else if next(iters[i]) then
        m ← get(iters[i]);
        metadata ← execute(m, ctx);
        if not Failed(metadata) then
            ctx.push(metadata);
            i ← i + 1;
    else
        ctx.pop();
        rewind(iters[i]);
        i ← i − 1;
return test
```

## 4.3. Experiment orchestration

When the planner is done generating benchmarking configuration tuples, it is necessary to prune unnecessary entries and make a schedule for the rest. There are multiple invariances in the benchmarking tasks to help pruning. For example, given the same AI model, software/hardware configuration, and similar problem size (of different problems), the throughput (in terms of FLOPS) can be comparable. Likewise, the precision evaluation should not be heavily impacted for the same model and problem on different software/hardware configurations. The executor should only pick significant tuples to maximize the diversity in measurements for all the metric dimensions, including model performance, scalability, generalization, and so on. Once the pruning is done, the scheduling problem concerns how to estimate the costs of each tuple, and efficiently pack them onto the hardware-task timeline.

```
@SoftwareDefinition
def TensorFlow(hw):
  tag = ':gpu'         if hw.UseCuda and hw.Arch == 'x64' else \
        ''             if not hw.UseCuda and hw.Arch == 'x64' else \
        ':gpu-ppc64le' if hw.UseCuda and hw.Arch == 'ppc64le' else \
        ':ppc64le'     if not hw.UseCuda and hw.Arch == 'ppc64le' else \
        Fail('unsupported hardware configuration')
  Docker('tensorflow/tensorflow' + tag)
```

**Fig. 4.** TensorFlow Software Configuration.

```
@ConverterDefinition
def AtomEmbedding(a: Atom) -> Tensor:
    return atom_embeddings[a.name]
@ConverterDefinition
def Concat(a: list[T]) -> Tensor:
    return Concat(a.map(Convert(T,Tensor)))

@ModelDefinition
def MLP(task, inputType, outputType, hiddenSize):
  Suggest(hiddenSize, 128, 256, 512, 1024)
  if task == Classify:
    f = Softmax(outputType.dims)
  elif task == Enhance:
    f = id
  # ...
  else: Fail("don't know how to solve")
  return Pipeline([
    Convert(inputType, Tensor),
    Linear(hiddenSize), Relu(),
    Linear(outputType.dims), f,
    Convert(Tensor, outputType)
  ])
```

**Fig. 5.** AI Model Definition.

## 5. Case study

Now we discuss the details of a particular use case, Molecular Dynamics (MD). Given the initial states of the atoms (position and velocity vectors), the problem asks for a prediction of the movement of the atoms. In practice, the problem is decomposed into the problem of force prediction (Molecular Mechanics), and the integrated force over time to compute new states. In particular, force prediction is achieved in multiple ways developed by the Molecular Dynamics research community. "Classical MD" employs empirical models to compute pairwise forces between atoms, and first-principle methods (AIMD) employ quantum mechanic methods as DFT [25] and CCSD(t) [26] to first predict the potential energy of the system, and then obtain the forces by computing the partial derivatives of the energy over atom positions.

The problem definition module is shown in Fig. 6. It consists of two phases. First, an AI model is trained to predict the potential energy of a system, guided by a Molecular Dynamics software package, such as ORCA or Gaussian. Then, the performance of the model is evaluated on a different set of atom configurations. Unlike traditional AI benchmarks that merely evaluate the output of a model, here we provide multiple fixtures, including both the energy prediction, and the position and velocity updates computed from the prediction. It is the flexibility of problem scripting that gives us the ability to model additional fixtures other than the energy, which can be extended to benchmark fields other than Molecular Dynamics, for example, Raman Spectroscopy. This is a typical "defined by setting" problem as we illustrated in Section 2, because although it reads multiple data points from input files, each data point is not fed into the AI model, but rather into a simulation software to compute data points for the AI model.

There are multiple ways to specify an AI model for this problem — namely, given a set of atoms (atom types, positions, and velocities), predict a single scalar energy value. One way is to implement an end-to-end energy prediction model [27,28]. The other way aims to capture the essence of the end-to-end solutions and let the system synthesize the whole model. One key insight of the aforementioned energy prediction models is that the atom configuration is permutation invariant, which means that the input should be modeled as a set of atoms, not a list. Therefore, our goal here is to enable the system to compose an AI model to honor this property and take advantage of existing building blocks. A possible solution is shown in Fig. 7, where the input is typechecked to be a list, and the module requires a submodule that can complete the specified task (prediction in the Molecular Dynamics context) to map the element type to the output type. The element-wise results are then summed to combine a permutation invariant output. This way, the system is able to pick up the modules we defined earlier, such as the atom embedding converter, and the MLP model for conducting element-wise prediction.

Now we discuss another benchmarking scenario, deep-learning-based electron microscopy image segmentation, which is becoming a popular topic in Biological Chemistry [29–31]. One of the main challenges in this topic is the scarce of training data, due to complex and costly data acquisition process. Given limited data, supervised deep-learning methods require heavy human intervention and may fail to generalize to unseen data [32,33]. One way to circumvent the data problem is to introduce semi-supervised deep-learning techniques, such as pre-training with high volume unlabeled data [34]. To support pre-training in the benchmarking system means that a model under evaluation should be able to carry a part of its internal states (weights) from one task to another, and adjust its computation graph accordingly. The problem definition should also evaluate the performance of the model given different amounts of training data, to test its sample efficiency. The code for this scenario is shown in Fig. 8.

### 5.1. Comparison to other benchmarking systems

As mentioned above, previous systems focus on a fixed set of test scenarios [19–21]. Additionally, the lack of declarative modules means that it is hard to share data between the benchmarking suite and external scientific computing software packages, which is crucial in scientific AI benchmarking. For example, the **Gradient** primitive in SAIBench allows a training pipeline to extract gradients from an external package, which is usually not exposed programmatically. The differences are shown in Table 4.

```
@ProblemDefinition
def MDSimulation():
    md = Require(MolecularDynamicsSoftware)
    # Training phase
    train_problems = [
        Input("ch2o.atoms").XYZ(),
        # ...
    ]
    for atoms in train_problems:
        for _ in range(200):
            e = md.PredictEnergy(atoms)
            Train.Predict(atoms, e)
            f = md.ComputeForces(atoms, e)
            atoms = md.ComputeNewStates(atoms, 1.0, f)
    # Testing phase
    test_problems = [
        Input("h2o.atoms").XYZ(),
        # ...
    ]
    for atoms in test_problems:
        atoms_golden = atoms
        atoms_pred = atoms
        for _ in range(200):
            e_g = md.PredictEnergy(atoms_golden)
            e_p = Model.Predict(atoms_pred)
            Test.Compare(e_g, e_p)
            f_g = md.ComputeForces(atoms_golden, e_g)
            pos = atoms_pred.map(lambda x: x.pos)
            f_p = Gradients(pos, e_p)
            atoms_golden = md.ComputeNewStates(atoms_golden, 1.0, f_g)
            atoms_pred = md.ComputeNewStates(atoms_pred, 1.0, f_p)
            Test.Compare(atoms_golden, atoms_pred)
```

**Fig. 6.** Molecular Dynamics Problem Definition.

```
@ModelDefinition
def PermutationInvariantModel(task, inputType, outputType):
    if not inputType.IsSet: Fail("Don't know how to solve")
    ety = inputType.ElementType
    m = Require(Model, task, ety, outputType)
    return Sum(Array(m, inputType.Size))
```

**Fig. 7.** Permutation Invariant Model Definition.

**Table 4**
Comparison to other benchmarking systems.

| | SAIBench | MLPerf | MLHarness |
|---|---|---|---|
| Focus | Different scientific tasks/criterion | Accuracy, system throughput | Scalability, MLCommon coverage |
| Modules | Declarative | Hard-coded | Markup |
| Test scenarios | Automatic discovery | Fixed | Fixed |

## 6. Discussion

We have elaborated on the methodology and the overview of the system design, yet we look forward to further development in the components. Brute-force enumeration of all possible test hyperparameters may not be feasible and while pruning can mechanically improve the situation, it is desirable that a particular problem module can suggest parameters suitable for a research field. More design work could be done to address model development and debugging needs, for example, to allow model validation in addition to training and testing. Python-based eDSL has its limitations, mostly due to the syntactic constraints of the language. To represent the modules more naturally, a programming language more geared towards scientific computing can be investigated [35].

Currently, SAIBench targets tractable scientific tasks, which are mechanical procedures that can be computed and measured. It is challenging to extend it to more creative scientific research activities because it would require the system to formally model the scientific concepts, and gain a deeper understanding of research topics, motivations, methodologies, and goals, and how various concepts interact with each other. Also, automated benchmarks require well-defined metrics, while open-ended scientific research ideas, in general, are hard to quantify.

Apart from type-based model composition, automatic AI model synthesizing given a particular problem definition is also a promising direction, given the advancement in AI-based code generation [36,37].

## 7. Conclusion

We have presented our definition of scientific AI benchmarking, which is an ensemble of scientific task definition, AI benchmarking, and system performance benchmarking. We have then presented our methodology for scientific AI benchmarking, with the key idea of decoupling and modularizing various components, automatically benchmarking sensible combinations. We have proposed a system design

```
@ModelDefinition
def PretrainSegmentModel(task, inputType, outputType, embsize):
    embType = Tensor(embsize)
    # base model
    m_base = Require(Model, Embedding, inputType, embType)
    if task == Pretrain:
        # pretrain decoder
        m_pretrain = Require(Model, Segment, embType, inputType)
        return Pipeline([ m_base, m_pretrain ])
    elif task == Segment:
        # task decoder
        m_task = Require(Model, task, embType, outputType)
        return Pipeline([ m_base, m_task ])
    else:
        Fail("don't know how to solve.")

@ProblemDefinition
def ElectronMicroscopySegmentation():
    pretrain_data = Input("CEM500K").Image()
    labeled_data = [
        [Input("cell0_slice1.img").Image(),
         Input("cell0_slice1.msk").Image()],
        # ...
    ]
    for unlabled in pretrain_data:
        Train.Pretrain(unlabled)
    train_data, test_data = RandomSplit(labeld_data, 0.9)
    train_stages = Chunks(train_data, 10)
    for i in range(10):
        train_current = sum(train_stages[0..i])
        for x,y in train_current:
            Train.Segment(x,y)
        for x,y in test_data:
            Test.Segment(x,y)
```

**Fig. 8.** Electron Microscopy Image Segmentation.

where the various modules are implemented with a domain-specific language for scientific AI computing. We have demonstrated that this design is flexible enough to support benchmarking different types of scientific tasks, defining AI models, deriving multiple metrics, combining metrics into ranking criteria, and configuring required hardware/software.

## Declaration of competing interest

Yatao Li reports financial support was provided by Microsoft Research Asia. Yatao Li reports administrative support was provided by Institute of Computing Technology Chinese Academy of Sciences. Jianfeng Zhan reports financial support was provided by Institute of Computing Technology Chinese Academy of Sciences.

## References

[1] A.N. Laboratory, AI for science report.URL https://publications.anl.gov/anlpubs/2020/03/158802.pdf.

[2] K. Albertsson, P. Altoe, D. Anderson, J. Anderson, M. Andrews, J.P.A. Espinosa, A. Aurisano, L. Basara, A. Bevan, W. Bhimji, D. Bona-corsi, B. Burkle, P. Calafiura, M. Campanelli, L. Capps, F. Carmi-nati, S. Carrazza, Y.-f. Chen, T. Childers, Y. Coadou, E. Coniavitis, K. Cranmer, C. David, D. Davis, A. De Simone, J. Duarte, M. Erd-mann, J. Eschle, A. Farbin, M. Feickert, N.F. Castro, C. Fitzpatrick, M. Floris, A. Forti, J. Garra-Tico, J. Gemmler, M. Girone, P. Glaysher, S. Gleyzer, V. Gligorov, T. Golling, J. Graw, L. Gray, D. Greenwood, T. Hacker, J. Harvey, B. Hegner, L. Heinrich, U. Heintz, B. Hoober-man, J. Junggeburth, M. Kagan, M. Kane, K. Kanishchev, P. Karpiński, Z. Kassabov, G. Kaul, D. Kcira, T. Keck, A. Klimentov, J. Kowalkowski, L. Kreczko, A. Kurepin, R. Kutschke, V. Kuznetsov, N. Köhler, I. Lako-mov, K. Lannon, M. Lassnig, A. Limosani, G. Louppe, A. Mangu, P. Mato, N. Meenakshi, H. Meinhard, D. Menasce, L. Moneta, S. Moort-gat, M. Neubauer, H. Newman, S. Otten, H. Pabst, M. Paganini, M. Paulini, G. Perdue, U. Perez, A. Picazio, J. Pivarski, H. Prosper, F. Psihas, A. Radovic, R. Reece, A. Rinkevicius, E. Rodrigues, J. Rorie, D. Rousseau, A. Sauers, S. Schramm, A. Schwartzman, H. Severini, P. Seyfert, F. Siroky, K. Skazytkin, M. Sokoloff, G. Stewart, B. Stienen, I. Stockdale, G. Strong, W. Sun, S. Thais, K. Tomko, E. Upfal, E. Usai, A. Ustyuzhanin, M. Vala, J. Vasel, S. Vallecorsa, M. Verzetti, X. Vilasís-Cardona, J.-R. Vlimant, I. Vukotic, S.-J. Wang, G. Watts, M. Williams, W. Wu, S. Wunsch, K. Yang, O. Zapata, Machine learning in high energy physics community white paper. URL http://arxiv.org/abs/1807.02876.

[3] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, M. Houston Prabhat, Exascale deep learning for climate analytics, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis,

[4] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Don-ner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J.-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, M. Riedmiller, Magnetic control of toka- mak plasmas through deep reinforcement learning 602 (7897) 414–419. http://dx.doi.org/10.1038/s41586-021-04301-9. URL https://www.nature.com/articles/s41586-021-04301-9.

[5] R.B. Neale, A. Gettelman, S. Park, C.-C. Chen, P.H. Lauritzen, D.L. Williamson, A.J. Conley, D. Kinnison, D. Marsh, A.K. Smith, F. Vitt, R. Garcia, J.-F. Lamarque, M. Mills, S. Tilmes, H. Morrison, P. Cameron-Smith, W.D. Collins, M.J. Iacono, R.C. Easter, X. Liu, S.J. Ghan, P.J. Rasch, M.A. Taylor, Description of the NCAR community atmosphere model (CAM 5.0) 289.

[6] J.S. Smith, R. Zubatyuk, B. Nebgen, N. Lubbers, K. Barros, A.E. Roit-berg, O. Isayev, S. Tretiak, The ANI-1ccx and ANI-1x data sets, coupled-cluster and density functional theory properties for molecules, Sci. Data 7 (1) 134. http://dx.doi.org/10.1038/s41597-020-0473-z. URL http://www.nature.com/articles/s41597-020-0473-z.

[7] L. Ruddigkeit, R. van Deursen, L.C. Blum, J.-L. Reymond, Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17, J. Chem. Inform. Model. 52 (11) 2864–2875. http://dx.doi.org/10.1021/ci300415d. URL https://pubs.acs.org/doi/10.1021/ci300415d.

[8] D.S. Marcus, T.H. Wang, J. Parker, J.G. Csernansky, J.C. Morris, R.L. Buckner, Open access series of imaging studies (OASIS): Cross-sectional MRI data in young, middle aged, nondemented, and demented older adults, J. Cogn. Neurosci. 19 (9) 1498–1507. http://dx.doi.org/10.1162/jocn.2007.19.9.1498. URL https://direct.mit.edu/jocn/article/19/9/1498/4427/Open-Access-Series-of-Imaging-Studies-OASIS-Cross.

[9] E. Weinan, J. Han, A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. 5 (4) 349–380. http://dx.doi.org/10.1007/s40304-017-0117-6. URL https://collaborate.princeton.edu/en/publications/deep-learning-based-numerical-methods-for-high-dimensional-parabo.

[10] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Computat. Phys. 378 686–707. http://dx.doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[11] F. Noé, Machine learning for molecular dynamics on long timescales, in: K.T. Schütt, S. Chmiela, O.A. von Lilienfeld, A. Tkatchenko, K. Tsuda, K.-R. Müller (Eds.), Machine Learning Meets Quantum Physics, Springer International Publishing, pp. 331–372, http://dx.doi.org/10.1007/978-3-030-40245-7_16.

[12] A. Mardt, L. Pasquali, H. Wu, F. Noé, VAMPnets for deep learning of molecular kinetics, Nature Commun. 9 (1) 5, http://dx.doi.org/10.1038/s41467-017-02388-1. URL https://www.nature.com/articles/s41467-017-02388-1.

[13] W. Jia, H. Wang, M. Chen, D. Lu, L. Lin, R. Car, W. E, L. Zhang, Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning Version: 1. arXiv:2005.00223. URL http://arxiv.org/abs/2005.00223.

[14] T. Hoefler, R. Belli, Scientific benchmarking of parallel computing sys- tems: Ttwelve ways to tell the masses when reporting performance re- sults, in: Proceedings of the International Conference for High Perfor- Mance Computing, Networking, Storage and Analysis, ACM, pp. 1–12, http://dx.doi.org/10.1145/2807591.2807644, URL https://dl.acm.org/doi/10.1145/2807591.2807644.

[15] E. Apra‘, E.J. Bylaska, W.A. de Jong, N. Govind, K. Kowalski, T.P. Straatsma, M. Valiev, H.J.J. van Dam, Y. Alexeev, J. Anchell, V. Anisi-mov, F.W. Aquino, R. Atta-Fynn, J. Autschbach, N.P. Bauman, J.C. Becca, D.E. Bernholdt, K. Bhaskaran-Nair, S. Bogatko, P. Borowski, J. Boschen, J. Brabec, A. Bruner, E. Cauët, Y. Chen, G.N. Chuev, C.J. Cramer, J. Daily, M.J.O. Deegan, T.H. Dunning, M. Dupuis, K.G. Dyall, G.I. Fann, S.A. Fischer, A. Fonari, H. Früchtl, L. Gagliardi, J. Garza, N. Gawande, S. Ghosh, K. Glaesemann, A.W. Götz, J. Ham-mond, V. Helms, E.D. Hermes, K. Hirao, S. Hirata, M. Jacquelin, L. Jensen, B.G. Johnson, H. Jónsson, R.A. Kendall, M. Klemm, R. Kobayashi, V. Konkov, S. Krishnamoorthy, M. Krishnan, Z. Lin, R.D. Lins, R.J. Littlefield, A.J. Logsdail, K. Lopata, W. Ma, A.V. Marenich, J. Martin del Campo, D. Mejia-Rodriguez, J.E. Moore, J.M. Mullin, T. Nakajima, D.R. Nascimento, J.A. Nichols, P.J. Nichols, J. Nieplocha, A. Otero-de-la Roza, B. Palmer, A. Panyala, T. Pirojsirikul, B. Peng, R. Peverati, J. Pittner, L. Pollack, R.M. Richard, P. Sadayappan, G.C. Schatz, W.A. Shelton, D.W. Silverstein, D.M.A. Smith, T.A. Soares, D. Song, M. Swart, H.L. Taylor, G.S. Thomas, V. Tipparaju, D.G. Truh-lar, K. Tsemekhman, T. Van Voorhis, Vázquez-Mayagoitia, P. Verma, O. Villa, A. Vishnu, K.D. Vogiatzis, D. Wang, J.H. Weare, M.J. Williamson, T.L. Windus, K. Woliński, A.T. Wong, Q. Wu, C. Yang, Q. Yu, M. Zacharias, Z. Zhang, Y. Zhao, R.J. Harrison, NWChem: Past, present, and future 152 (18) 184102. http://dx.doi.org/10.1063/5.0004997. URL http://aip.scitation.org/doi/10.1063/5.0004997.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, PyTorch: An imperative style, high- performance deep learning library, in: Advances in Neural Information Processing Systems, Vol. 32, Curran Associates, Inc., URL https://papers.nips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

[17] M. Brehm, SANscript – A scientific algorithm notation language. URL https://brehm-research.de/sanscript.php.

[18] M.D. Wilkinson, M. Dumontier, I.J. Aalbersberg, G. Appleton, M. Ax-ton, A. Baak, N. Blomberg, J.-W. Boiten, L.B. da Silva Santos, P.E. Bourne, J. Bouwman, A.J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C.T. Evelo, R. Finkers, A. Gonzalez-Beltran, A.J.G. Gray, P. Groth, C. Goble, J.S. Grethe, J. Heringa, P.A.C. ’t Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S.J. Lusher, M.E. Martone, A. Mons, A.L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M.A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Vel-terop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, B. Mons, The FAIR guiding principles for scientific data management and stewardship, Sci. Data 3 (1) 160018. http://dx.doi.org/10.1038/sdata.2016.18. URL https://www.nature.com/articles/sdata201618.

[19] W. Gao, C. Luo, L. Wang, X. Xiong, J. Chen, T. Hao, Z. Jiang, F. Fan, M. Du, Y. Huang, F. Zhang, X. Wen, C. Zheng, X. He, J. Dai, H. Ye, Z. Cao, Z. Jia, K. Zhan, H. Tang, D. Zheng, B. Xie, W. Li, X. Wang, J. Zhan, Aibench: Towards scalable and comprehensive datacenter AI benchmarking, in: C. Zheng, J. Zhan (Eds.), Benchmarking, Measuring, and Optimizing, Vol. 11459, in: Lecture Notes in Computer Science, Springer International Publishing, pp. 3–9, http://dx.doi.org/10.1007/978-3-030-32813-9_1, URL http://link.springer.com/10.1007/978-3-030-32813-9 1.

[20] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Pat-terson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, A. Ike, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, G. Ma, D. Narayanan, T. Ogun-tebi, G. Pekhimenko, L. Pentecost, V.J. Reddi, T. Robie, T.S. John, T. Tabaru, C.-J. Wu, L. Xu, M. Yamazaki, C. Young, M. Zaharia, MLPerf training benchmark 14.

[21] Y.-H. Chang, J. Pu, W.-m. Hwu, J. Xiong, MLHarness: A scalable benchmarking system for ML Commons, BenchCouncil Trans. Benchmarks, Standards Eval. 1 (1) 100002. http://dx.doi.org/10.1016/j.tbench.2021.100002. URL https://www.sciencedirect.com/science/article/pii/S2772485921000028.

[22] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in PyTorch 4.

[23] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) 2278–2324. http://dx.doi.org/10.1109/5.726791. URL http://ieeexplore.ieee.org/document/726791/.

[24] J. Thiyagalingam, M. Shankar, G. Fox, T. Hey, Scientific machine learn- ing benchmarks.URL http://arxiv.org/abs/2110.12773.

[25] R. Haunschild, A. Barth, B. French, A comprehensive analysis of the history of DFT based on the bibliometric method RPYS, J. Cheminform. 11 (1) 72, http://dx.doi.org/10.1186/s13321-019-0395-y.

[26] H.G. Kümmel, A biography of the coupled cluster method 17 (28) 5311–5325, http://dx.doi.org/10.1142/S0217979203020442. URL https://www.worldscientific.com/doi/abs/10.1142/S0217979203020442.

[27] J. Han, L. Zhang, R. Car, W. E, Deep potential: A general representation of a many-body potential energy surface, Commun. Computat. Phys. 23 (3). arXiv:1707.01478, http://dx.doi.org/10.4208/cicp.OA-2017-0213. URL http://arxiv.org/abs/1707.01478.

[28] O.T. Unke, M. Meuwly, PhysNet: A neural network for predicting energies, forces, dipole moments and partial charges, J. Chem. Theory Computat. 15 (6) 3678–3693. arXiv:1902.08408, http://dx.doi.org/10.1021/acs.jctc.9b00181. URL http://arxiv.org/abs/1902.08408.

[29] E. Gómez-de Mariscal, M. Maška, A. Kotrbová, V. Pospíchalová, P. Mat-ula, A. Munõz-Barrutia, Deep-learning-based segmentation of small extracellular vesicles in transmission electron microscopy images, Sci. Rep. 9 (1) 13211. http://dx.doi.org/10.1038/s41598-019-49431-3. URL https://www.nature.com/articles/s41598-019-49431-3.

[30] L. von Chamier, R.F. Laine, J. Jukkala, C. Spahn, D. Krentzel, E. Nehme, M. Lerche, S. Hernández-Pérez, P.K. Mattila, E. Karinou, S. Holden, A.C. Solak, A. Krull, T.-O. Buchholz, M.L. Jones, L.A. Royer, C. Leterrier, Y. Shechtman, F. Jug, M. Heilemann, G. Jacquemet, R. Henriques, Democratising deep learning for microscopy with ZeroCostDL4Mic, Nature Commu. 12 (1) 2276. http://dx.doi.org/10.1038/s41467-021-22518-0. URL https://www.nature.com/articles/s41467-021-22518-0.

[31] J.M. Ede, Deep learning in electron microscopy, Mach. Learning: Sci. Technol. 2 (1) 011004. http://dx.doi.org/10.1088/2632-2153/abd614.

[32] S.M. Plaza, J. Funke, Analyzing image segmentation for connectomics, Front. Neural Circ. 12, 102. DOI: http://dx.doi.org/10.3389/fncir.2018.00102. URL https://www.frontiersin.org/article/10.3389/fncir.2018.00102/full.

[33] J.W. Lichtman, H. Pfister, N. Shavit, The big data challenges of connectomics, Nature Neurosci. 17 (11) 1448–1454. http://dx.doi.org/10.1038/nn.3837. URL http://www.nature.com/articles/nn.3837.

[34] R. Conrad, K. Narayan, CEM500K, a large-scale heterogeneous unlabeled cellular electron microscopy image dataset for deep learning, eLife 10 e65894, eLife Sciences Publications, Ltd. DOI: http://dx.doi.org/10.7554/eLife.65894.

[35] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V.B. Shah, W. Tebbutt, A differentiable programming system to bridge machine learning and scientific computing. URL http://arxiv.org/abs/1907.07587.

[36] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tu-fano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S.K. Deng, S. Fu, S. Liu, CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. URL http://arxiv.org/abs/2102.04664.

[37] D. Peng, S. Zheng, Y. Li, G. Ke, D. He, T.-Y. Liu, How could neural networks understand programs? in: Proceedings of the 38th International Conference on Machine Learning, PMLR, pp. 8476–8486, URL https://proceedings.mlr.press/v139/peng21b.html.