Electronic Supplementary Material (ESI) for Digital Discovery.


# Supplementary Information: By how much can closed loop frameworks accelerate computational materials discovery?

Lance Kavalsky,[1,*] Vinay I. Hegde,[2,*] Eric Muckley,[2] Matthew S. Johnson,[3] Bryce Meredig,[2,†] and Venkatasubramanian Viswanathan[1,‡]

[1] *Carnegie Mellon University, Pittsburgh, PA 15213*

[2] *Citrine Informatics, Redwood City, CA 94063*

[3] *Massachusetts Institute of Technology, Cambridge, MA 02139*

## I. EFFECTIVENESS OF ACQUISITION FUNCTIONS

Here, we expand the set of acquisition functions and compare their effectiveness for the three sequential learning (SL) tasks considered in the main text: (a) finding candidates in the target window, (b) surfacing candidates of high quality, i.e., close to the target window, and (c) building minimal datasets for training ML models that are close in accuracy to those trained on the full dataset.

We define a new baseline for acquisition functions, a "space-filling" approach, that works as

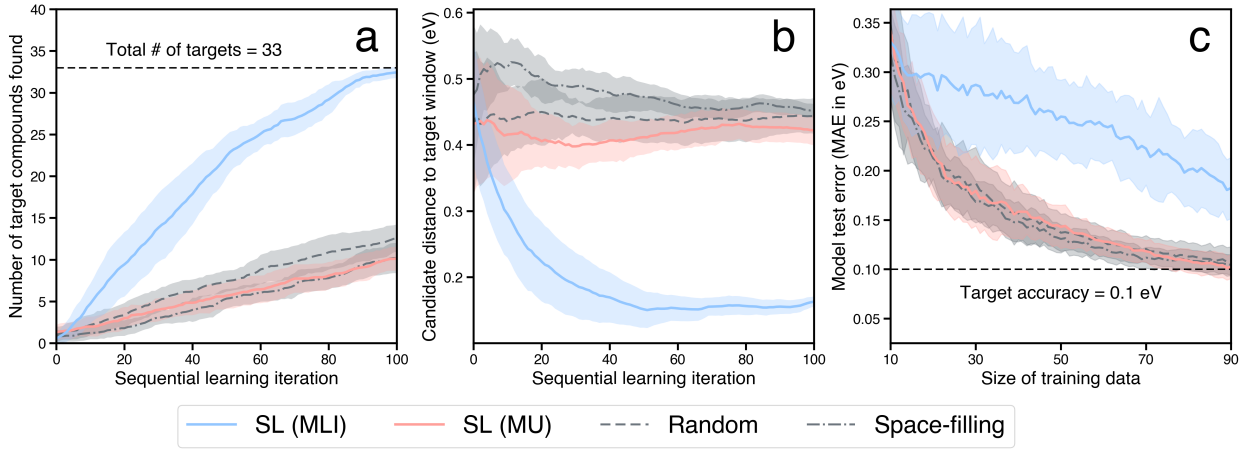


FIG. S1. A comparison of random search vs sequential learning (SL)-driven approach to find new bimetallic catalysts with a target property. (a) Overall, the SL-driven approach identifies all the 33 target candidates in the dataset within 100 iterations, ∼3× faster than random search and a space-filling-based acquisition function. (b) Candidates surfaced via SL lie much closer to the target window on average, when compared to those chosen via random search or space-filling. (c) An SL-driven approach can help identify a much smaller number of examples that can be used to train ML surrogates to a desired accuracy, at a fraction of the overall dataset size. Here, the overall dataset has ∼300 candidates, and an ML model trained on only ∼25% of the candidates chosen via a SL-driven maximum uncertainty (MU)-based approach achieves the target accuracy. Notably, the other purely-exploratory acquisition functions considered here, random and space-filling, perform as well as MU for building datasets. In each case, the shaded region in the plots represent variation in the reported quantities estimated over 20 independent trials.


* These authors contributed equally to this work

† bryce@citrine.io

‡ venkvis@cmu.edu

follows: (1) Start with a randomly-chosen initial training set (e.g., ∼10, used previously), (2) Build ML models, (3) From the design space of candidates, choose one that is *farthest* from all the training examples (measured using Euclidian distance of each candidate, in the 145-dimensional Magpie feature vector-space, to the closest example in the current training set) to augment the training set in each SL iteration, (4) Re-train ML models, (5) Perform steps (1)–(5) for a predetermined number of iterations or until the termination condition is met. This space-filling acquisition strategy performs similarly to the random (uniform) acquisition strategy in finding candidates, candidate quality, as well as for building datasets for ML surrogates (see FIG. S1).

Expectedly, MU, random, and space-filling perform worse than MLI for finding target candidates and surfacing quality candidates (FIG. S1a,b), as both are SL tasks that benefit from balancing exploration with exploitation. On the other hand, the space-filling and random acquisition strategies perform as well as the MU-based one for building training datasets (and all three exploratory acquisition strategies perform better than MLI), demonstrating that the three approaches are nearly-equally effective for pure exploration (FIG. S1c). This is consistent with previous reports comparing model accuracy as a function of SL iteration using similar acquisition functions [1].

In terms of picking an acquisition strategy specifically for constructing training datasets, there is additional nuance to consider related to cost. While the space-filling and random acquisition functions do not require ML model building in each SL iteration (and are thus faster and more economical compared to MU), the effectiveness of the three approaches is seen to vary based on the data distribution in the particular design space of interest [1]. This can be the subject of future investigation.

## II. HUMAN LAGTIME MODEL

When defining our human lag model, we invoke the following assumptions:

1. 3 windows of time:

   (a) Researcher at work (9am-5pm): checks on job every couple of hours; average lag of 1 hour.

(b) Researcher (partially) away from work (5pm-11pm): checks on job at the end of the window; average lag of 3 hours.

(c) Researcher (completely) away from work (11pm-9am): checks on job at the end of window; average lag of 5 hours

2. Uniform distribution of when jobs will finish through a week:

(a) Probability of job finishing during the week = 5/7: researcher checks on job according to 1, above.

(b) Probability of job finishing during the weekend = 2/7: researcher checks on job once during the weekend; average lag of 24 hours.

## III. SURROGATE ACCURACY

The size of the unexplored design space shrinks as the simulated sequential learning (SL) progresses, i.e., the number of candidates in the full dataset that the model has not "seen" yet keeps continuously decreasing. So surrogate model accuracy estimates derived using model predictions over the entire unexplored design space (the test set) in each SL iteration can be affected by the continuously diminishing test set size. In order to mitigate this effect of test set size on the surrogate model accuracy estimates, we employ a bootstrapping approach to keep the test set size fixed in each SL iteration. At each SL iteration, 20 "bootstrap test samples" are generated from the full unexplored design space. Each of the 20 bootstrap test samples are generated by randomly sampling, with replacement, 100 candidates from the unexplored design space. For each of the 20 bootstrap test samples (with 100 candidates each) we calculate the mean absolute error (MAE). Finally, we run 20 independent trials of the entire simulated SL pipeline. The accuracy of the surrogate model at a given SL iteration is then defined by the mean and standard deviation of the MAEs of the bootstrap test samples (generated at that SL iteration) aggregated over the 20 independent trials. We use this final mean MAE of the surrogate model as the accuracy metric of interest with a target value of 0.1 eV.

## IV. SUMMARY OF ACCELERATION FACTORS

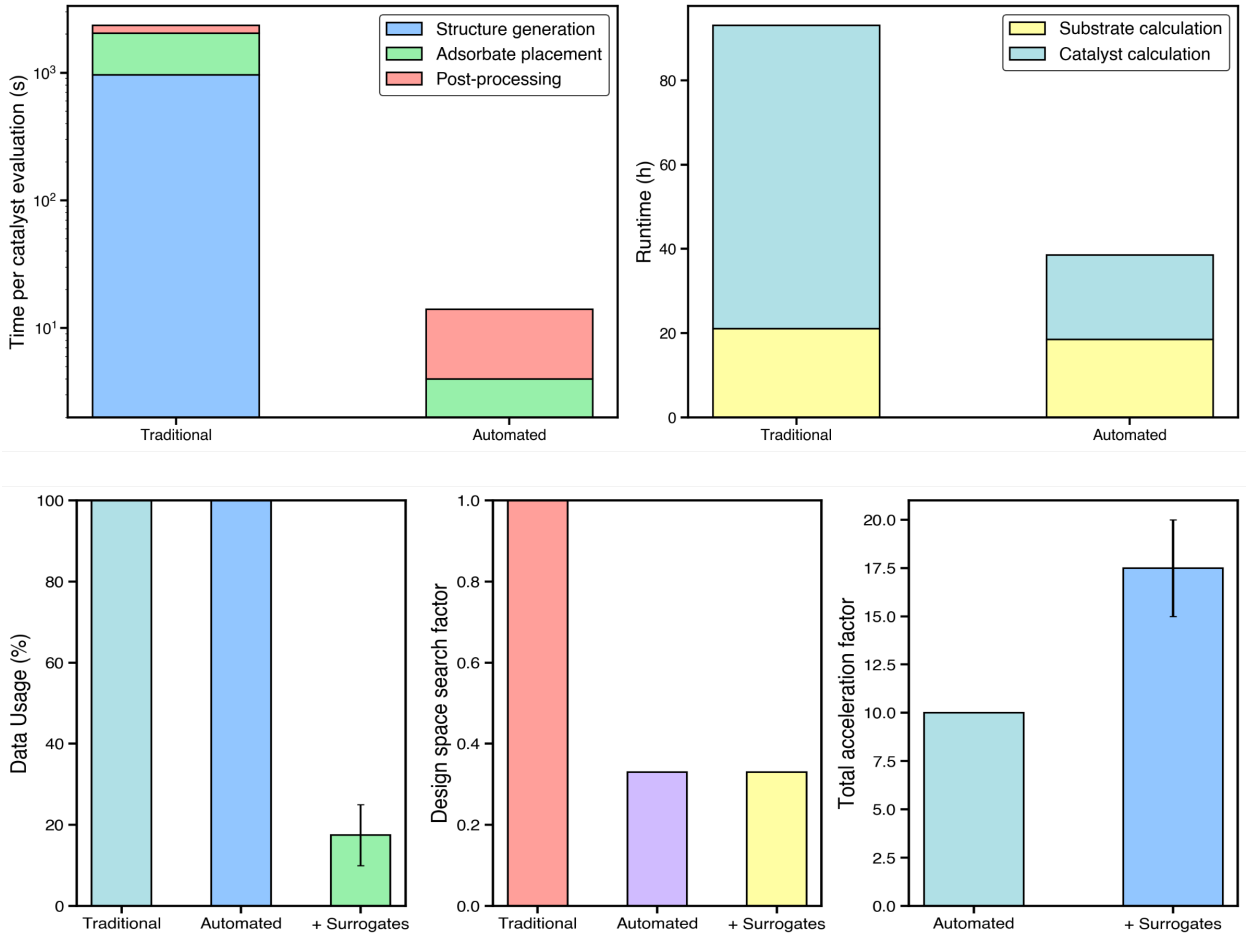


FIG. S2. Acceleration factors visualized as bar plots using the data from Table III of the main manuscript. The component-wise analysis highlighted here can be used to make informed decisions when designing closed-loop frameworks of varying topologies.

## V. DATA AND SCRIPTS FOR REPRODUCIBILITY

All data and Python scripts required to perform the analysis presented in this work are made available via the GitHub repository at https://github.com/aced-differentiate/closed-loop-acceleration-benchmarks. The repository is organized as follows:

1. data/

   - `benchmark_calculations_record.xlsx`: Excel spreadsheet containing a record of DFT calculations, associated raw timestamps, and a tabulation of the acceleration estimates.
   - bimetallic_catalysts_dataset/
     - `ma_2015_bimetallics_raw.json.gz`: Dataset of bimetallic alloys for CO2 reduction, in the Physical Information File (PIF) format, obtained from Dataset 153450 on Citrination.
       Original data source: "Machine-Learning-Augmented Chemisorption Model for CO2 Electroreduction Catalyst Screening", Ma et al., *J. Phys. Chem. Lett.* **6** 3528-3533 (2015). DOI: 10.1021/acs.jpclett.5b01660
     - `transform.py`: Python script for converting from the PIF format into tabular data.
     - `bimetallics_data.csv`: Bimetallics catalysts dataset mentioned above in a tabular format.
   - runtime_geometries/
     "Chemically-informed" and naive structures and settings in the form of `ase.traj` files, corresponding to the discussion surrounding FIG. 3 in the main text. The files can be read using ASE package (using `ase.io.read`).

2. scripts/

   - `human_lagtime.py`: Script for estimating human lagtime in job management, calculated using a Monte Carlo sampling method.
   - `sequential_learning.py`: Script for running multiple independent trials of sequential learning (SL) and recording a history of training examples, model predictions and prediction uncertainties.

If run as-is, the script performs 20 independent trials of 100 SL iterations to optimize the `binding_energy_of_adsorbed` property in the bimetallic catalysts dataset mentioned above, using four acquisition functions (results from each recorded separately): random, maximum likelihood of improvement (MLI), maximum uncertainty (MU), and space-filling.

- `plot_acceleration_from_sequential_learning.py`: Script to aggregate results from the `sequential_learning.py` script, calculate and plot statistics related to acceleration from SL over a baseline.

  If run as-is, the script reproduces the 3-paneled FIG. 5 in the main text.

- `plot_acceleration_from_sequential_learning__ALL_ACQ.py`: Similar to the previous script; plots and compares statistics from all acquisition functions considered (MLI, MU, random, space-filling) for all SL tasks.

  If run as-is, the script reproduces the 3-paneled FIG. S1 in the Supplementary Information.

- `plot_levels_of_automation.py`: Script for plotting the cumulative time of executing the DFT pipeline at varying levels of automation.

  If run as-is, the script reproduces the bottom panel from FIG. 2 in the main text.

## A. Running the scripts

The required packages for executing the scripts are specified in `requirements.txt`, and can be installed in a new environment (e.g. using conda) as follows:

```
$ conda create -n accel_benchmarking python=3.10
$ conda activate accel_benchmarking
$ pip install -r requirements.txt
```

The scripts are all in python, and can be run from the command line. For example:

```
$ cd scripts
$ python sequential_learning.py
```

[1] C. K. Borg, E. S. Muckley, C. Nyby, J. E. Saal, L. Ward, A. Mehta, and B. Meredig, Quantifying the performance of machine learning models in materials discovery, Digital Discovery **2**, 327 (2023).