EnzyHTP Computational Directed Evolution with Adaptive Resource Allocation

Qianzhen Shao[1], Yaoyukun Jiang[1] and Zhongyue J. Yang[1-5,*]

[1]*Department of Chemistry, Vanderbilt University, Nashville, Tennessee 37235, United States*

[2]*Center for Structural Biology, Vanderbilt University, Nashville, Tennessee 37235, United States*

[3]*Vanderbilt Institute of Chemical Biology, Vanderbilt University, Nashville, Tennessee 37235, United States* [4]*Data Science Institute, Vanderbilt University, Nashville, Tennessee 37235, United States* [5]*Department of Chemical and Biomolecular Engineering, Vanderbilt University, Nashville, Tennessee 37235, United States*

ABSTRACT: Directed evolution facilitates enzyme engineering via iterative rounds of mutagenesis. Despite the wide applications of high-throughput screening, building "smart libraries" to effectively identify beneficial variants remains a major challenge in the community. Here, we developed a new computational directed evolution protocol based on EnzyHTP, a software we have previously reported to automate enzyme modeling. To enhance the throughput efficiency, we implemented an adaptive resource allocation strategy that dynamically allocates different types of computing resources (e.g., GPU/CPU) based on the specific need of an enzyme modeling sub-task in the workflow. We implemented the strategy as a Python library and tested the library using fluoroacetate dehalogenase as a model enzyme. The results show that comparing to fixed resource allocation where both CPU and GPU are on-call for use during the entire workflow, applying adaptive resource allocation can save 87% CPU hours and 14% GPU hours. Furthermore, we constructed a computational directed evolution protocol under the framework of adaptive resource allocation. The workflow was tested against two rounds of mutational screening in the directed evolution experiments of Kemp eliminase with a total of 184 mutants. Using folding stability and electrostatic stabilization energy as computational readout, we reproduced three out of the four experimentally-observed target variants. Enabled by the workflow, the entire computation task (i.e., 18.4 μs MD and 18,400 QM single point calculations) completes in three days of wall clock time using ~30 GPUs and ~1000 CPUs.

Keywords: adaptive resource allocation, high throughput simulation, Python library, computational directed evolution

1. Introduction

Directed evolution has been widely employed to expand and enhance enzymes' catalytic capability via iterative rounds of screening and selecting of enzyme variants.[1, 2] Despite significant achievements in enzyme engineering, directed evolution faces big challenges in screening because of the enormously large combinatorial mutational space. Specifically, due to unknown sequence-function relationship, each round of screening may take substantial cost of time and resource. The outcome from the screening is also uncertain. Although high-throughput screening techniques have been used to facilitate directed evolution,[3] designing readout for the function of enzyme mutants is highly case dependent. To accelerate the process of mutational screening, computational strategies have been employed to mechanistically infer beneficial mutations,[4-8] identify hotspots,[9-11] reveal structure/sequence-function relationships[12-15], and build data-driven models for predictive biocatalysis.[16] These foundational efforts significantly broaden the scope of which functional enzyme variants can be understood, predicted and designed, but how to build a general-purpose virtual screening platform for biocatalyst discovery remains an open question.

Previously, our lab developed EnzyHTP[17] as a high-throughput enzyme modeling tool to automate the entire life cycle of enzyme modeling, including structural preparation, mutagenesis, molecular dynamic (MD), and quantum mechanics (QM) calculations. Enabled by EnzyHTP, here we further developed a computational directed evolution protocol that enables hundreds of mutants to be virtually screened based on their stability and capability of stabilizing the breaking bond in the rate-limiting transition state. Notably, prior to our work, the Kamerlin lab has pioneered the development of software, CADEE,[18] for semi-automatic screening of enzyme variants. CADEE specializes in free energy calculations using empirical valence-bond (EVB), which is different

from the current work that emphasizes the construction of a generic computational enzyme engineering platform.

Central to the new development and implementation is an adaptive resource allocation strategy that accelerates the high-throughput computation via dynamically requesting and distributing computing nodes for each sub-task in the high-throughput modeling workflow. Here, resource allocation refers to the process of managing different types of computing resources (e.g., CPU or GPU) for different tasks (e.g., QM or MD simulations). In high-performance computing (HPC) clusters, large amount of computing resources can be wasted if a fixed number of CPU and GPU nodes are requested based on the resource need of the most computationally demanding sub-tasks. Besides resource waste, the fixed allocation scheme can also significantly delay the completion of the entire workflow.

The strategy of adaptive resource allocation was implemented as a Python application programming interface, known as adaptive resource manager (ARMer). Based on EnzyHTP,[17] we benchmarked the resource and time consumption of ARMer against fixed resource allocation scheme in the task of modeling fluoroacetate dehalogenase (FAcD)[19-23]. Furthermore, we constructed a computational directed evolution protocol to help screen for rate-enhancing enzyme mutants. We tested the workflow for two rounds of directed evolution screening of Kemp eliminase variants. Enabled by adaptive resource allocation, the computational directed evolution protocol offers a tool for accelerating the process of screening and selecting beneficial enzyme variants for biocatalytic uses.

2. Design and Implementation

2a. The computational directed evolution protocol. We constructed the computational directed evolution protocol using modular python functions in EnzyHTP (see Supporting Information .zip). The protocol consists of four steps. First, the workflow converts a list of user-provided starting variants into structural models using structural preparation module. The file format of these structural models is compatible with the downstream molecular simulations. Second, for each starting variant, the workflow randomly generates a certain number of mutants (defined by the user) to form the mutant library for the subsequent computational screening. Third, the workflow computes thermostability score for all mutants in a high-throughput fashion using cartesian_ddg in Rosetta[24, 25]. As an independent task, the workflow also calculates electrostatic stabilization energy, ΔG_{elec},[12, 26] for all mutants via performing 100 ns MD simulation in AMBER[2] and QM single-point calculations (i.e., PBE1PBE/def2-SVP) in Gaussian 16[27] using 100 snapshots evenly extracted from the MD trajectory. Specifically, the ΔG_{elec} is derived from the dot product between the reacting bond dipole (by wavefunction analysis[28]) and the electric field strength of the enzyme (calculate by Coulomb's law with MM charges). Fourth, the workflow filters out 80% of the mutants that are predicted to be thermally unstable relative to the wild type (i.e., positive thermostability score). The remaining mutants are ranked based on their ΔG_{elec} values. Notably, completing the above-mentioned four steps concludes one round of computational directed evolution. A certain number of mutants (defined by the user) can be used as starting variants to initiate a new round of directed evolution.

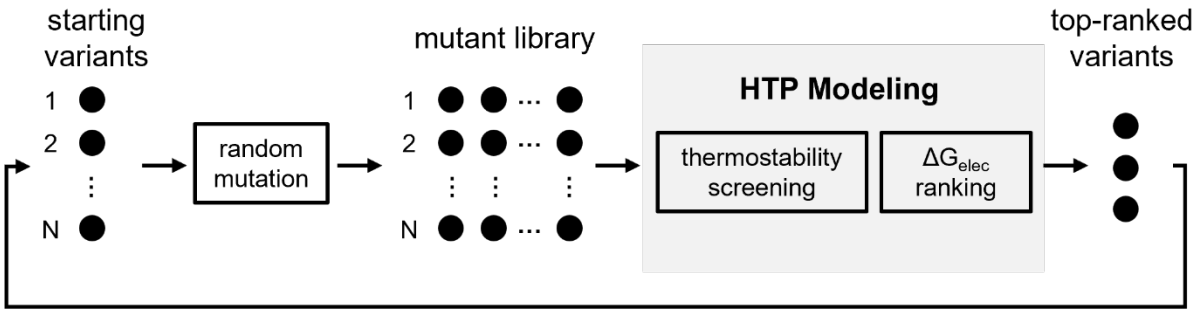

Figure 1. The design architecture of computational directed evolution protocol. Each black dot represents an enzyme variant.

2b. Adaptive resource manager (ARMer). To properly manage resource allocations for computational screening tasks, the adaptive resource allocation strategy employs a "workflow script" that runs a single-CPU thread to manage sub-tasks for the entire high-throughput workflow. Using commands implemented in the ARMer Python library, the workflow script configures, submits, and monitors new jobs in HPC clusters that pertain to the actual need of computing resources in a sub-task of the workflow. This is in sharp contrast to the fixed resource allocation scheme where maximal computing resources are requested.

The ARMer Python library consists of two classes: the Job class and HPC class (Supporting Information, Figure S1). The Job class (called ClusterJob in the code) defines properties and functions that are associated with job configuration, submission, and dynamic monitoring of job completion (Figure 2). The HPC class (subclasses of ClusterInterface in the code) supports the Job class with properties and functions to mediate shell input/output in user's local HPC where ARMer is deployed. This "plug-in" interface design allows easy support for any new HPC. Similar to *subprocess.run* in the Python standard library, ARMer library enables the "workflow script" to run

shell commands on other computational nodes – these commands are wrapped in the job scripts in the HPC clusters (see more discussion in the Supporting Information, Figure S1 and Text S1).

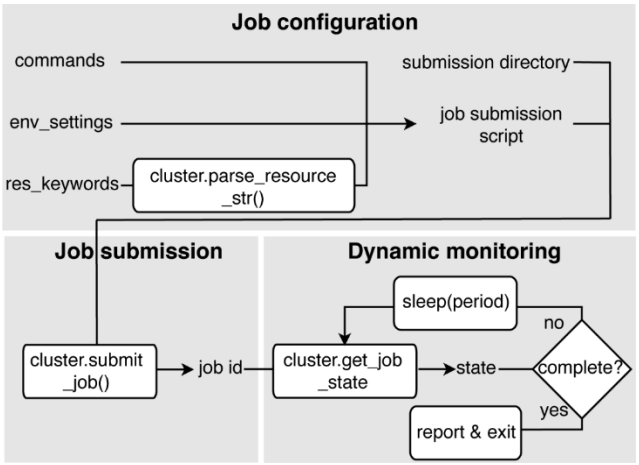

Figure 2. Variables and functions for job configuration, submission, and dynamic monitoring defined in the Job class.

With the job object instantiated, a job script for the required task can be generated (Supporting Information, Figure S3) and then submitted by the *submit()* method (Figure 2 and Supporting Information, Figure S2). Notably, the format of the job script, the submission commands, and other HPC-dependent information are obtained from the HPC class object that is instantiated and passed to the *cluster* argument described above. Once the job has been submitted, a job ID is added to the object by the function. By tracing the job ID, the "workflow script" can monitor the status of a job object in the queue, and mediate the status by killing, holding, or releasing the job (Figure 2). Notably, the capability of dynamically monitoring the job completion status is vital to high-throughput modeling workflow. This is because the workflow involves multiple different types of simulation sub-tasks that must be sequentially operated.

Two methods have been implemented to achieve dynamic monitoring – they are: *wait_to_end()*and *wait_to_array_end()* method. The *wait_to_end()* method checks the status of a job in the job queue with a certain period of time (i.e., every 30 seconds) and exits upon the detection of messages that indicate job completion, error, or cancellation. The *wait_to_array_end()* method takes multiple job objects and submits them in one job array. Similarly, the method monitors the status of all jobs in the array regularly, and dynamically append new jobs to the array up to the maximal capacity (i.e., array size).

Implementation of ARMer in EnzyHTP enables an *if_cluster_job* option for all computationally insensitive functions (like MD, QM and free energy calculation). With this option turned on, the computational directed evolution protocol described in the Section 2a can achieve adaptive resource allocation.

3. Results and Discussion

3a. Test of adaptive resource allocation using fluoroacetate dehalogenase. We employed fluoroacetate dehalogenase (FAcD)[19-23] as a model system and conducted a high-throughput workflow of enzyme simulations using EnzyHTP.[17] The workflow consists of four sequential sub-tasks, namely, 1) mutant structure construction, 2) 100 ns MD simulation, 3) 100 QM cluster calculation, and 4) post-analysis. (See details in Supporting Information Text S2 Notably, the model system and workflow have been applied in our previous work to test the throughput capability of EnzyHTP.[17] However, unlike the previous workflow that samples 100 variants with minimalist resource cost, the current workflow only tests one enzyme variant (i.e, K83D, Supporting Information, Figure S4) with MD and QM simulations set up to meet resource demand in actual computational research (Supporting Information .zip).

We benchmarked the resource and time consumption of the workflow for two strategies: fixed resource allocation versus adaptive resource allocation, on our local HPC at Vanderbilt, i.e., advanced computing center for research and education (ACCRE). Using fixed resource allocation, all computing resources (i.e., 1 GPU and 8 CPU) involved in the workflow are requested by a single submission script in one shot. Figure 3 shows the simulation type, time cost (i.e., vertical axis), and resource demand (i.e., GPU in orange and CPU in blue) associated with each sub-task in the workflow. Both CPU and GPU are requested for the entire workflow (Figure 3). However, resource waste is observed. Specifically, in the ~14 hours of MD simulation, only one GPU is used but the CPUs are primarily in idle mode (time-waste: ~8x14 = ~112 CPU hours); in the 3 hours of QM calculations, CPUs are used but the GPU is not (time-waste: 3 GPU hours). In the minimization and post-analysis sub-tasks, CPU and GPU are also not exploited, albeit the resource cost is trivial. Apparently, with fixed resource allocation, the collective resource waste is significant.

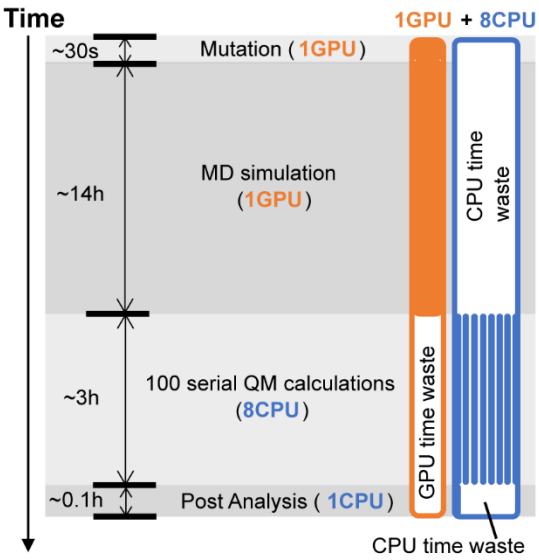

Figure 3. The high-throughput workflow of FAcD modeling using fixed resource allocation strategy, in which 1 GPU (in orange) and 8 CPUs (in blue) are requested in the beginning of the

workflow. The type of modeling sub-tasks, time cost, and resource consumption are noted on the Figure.

Using adaptive resource allocation, a 96-hour wall-clock time, single CPU job is submitted that operates a Python "workflow script" to allocate resources for sequential sub-tasks involved in the workflow (Figure 4). The "workflow script" manages the sub-tasks using commands implemented in the ARMer library (detailed in the Design and Implementation section). Compared to fixed allocation strategy that directly executes sub-tasks using the allocated CPU or GPU, this workflow script configures resource-demanding sub-tasks (i.e., need >1 CPU or ≥1 GPU) in a new job script and then submits the job to the queue (i.e., setting *if_cluster_job* = 'True' in the code).

In the MD simulation sub-task, the workflow script configures shell commands that run AMBER[29] simulations in a job script along with the GPU request and environment settings. The workflow script then submits the MD job and regularly monitors the completion status of the job. After confirming the completion of MD, the workflow script will continue operating the QM calculation sub-task in the workflow. Since the 100 QM calculations are independent, the workflow script can submit multiple QM jobs (8 CPU each) simultaneously to the job array so that they can run in parallel up to the size limit of job array (i.e., 25 jobs) in local HPC cluster (Figure 4). New jobs will be submitted once the "workflow script" detects open slots on the array (see discussion of parallel computing using Python subprocess module, Supporting Information Text S3). With an array size of 25 jobs, one would expect an ideal time acceleration by a factor of 25 given the ideal condition of no job queueing time (i.e., ~2.4 hours as compared to 60 hours with all job running serially). Overall, with adaptive resource allocation, the resource waste can be minimized.

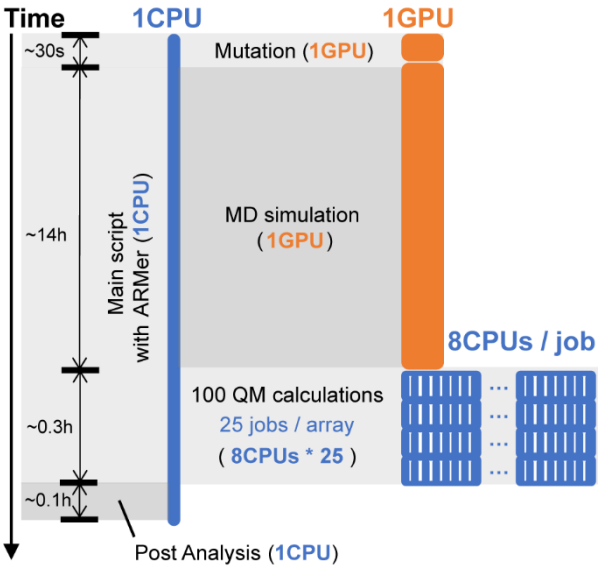

Figure 4. The high-throughput workflow of FAcD modeling using adaptive resource allocation strategy, in which a "workflow script" the runs on a single-CPU thread operates the modeling sub-tasks (i.e., mutation, MD, and QM) by configuring, submitting, and monitoring new job scripts. The MD job requests 1 GPU (in orange) and each QM job 8 CPUs (in blue). To submit and run individual QM calculations in parallel, a job array with a size of 25 is employed. The type of modeling sub-tasks, time cost, and resource consumption are noted on the Figure.

We compared the computational cost (Supporting Information, Figure S5) and wall clock time (Supporting Information, Figure S6) for different sub-tasks of the workflow using fixed resource allocation (in red) versus adaptive resource allocation (in green) on ACCRE. In contrast, fixed resource allocation involves 14% resource waste in GPU-based MD simulation and 87% in CPU-based QM calculations, while adaptive resource allocation makes full use of these computing resources in every sub-task of the workflow (Supporting Information, Figure S5). For wall clock time, both strategies differ most significantly in the QM calculation sub-task (Supporting Information, Figure S6). Enabled by parallel submission of individual QM jobs to a job array (i.e.,

25 jobs per array), adaptive resource allocation strategy completes 100 QM calculations by 0.35 hours. With fixed resource allocation (i.e., 8 CPUs), the calculations are completed by 2.8 hours. With larger QM system size and higher theory level, the QM resource consumption will be more significant. As such, the waste of resource and time for fixed resource allocation is expected to inflate.

With new jobs generated, submitted, and monitored on the fly by adaptive resource allocation strategy, the resource waste observed by fixed resource allocation is no long expected because resources are requested and distributed based on the need of individual sub-task in the workflow. We should note that one potential caveat of adaptive resource allocation strategy is the time spent for queueing. In our test, the total amount of time spent for job queueing is about 12 minutes, which is relatively trivial. On extremely crowded HPC, we assume job queueing time to be much longer. The fixed resource allocation scheme suffers less from the queueing, but the overall cost of time and resource is substantial.

Besides being resource-efficient, adaptive resource allocation scheme also prevents conflict of software environment setting. Under the adaptive resource allocation scheme, the workflow script can customize specific environment setting based on the requirement of an individual job. In contrast, the fixed resource allocation scheme will have to load all environmental variables in the beginning of the submission job, which can cause conflict for different software as the workflow proceeds.

3b. Test of the computational directed evolution protocol. We employed Kemp eliminase (KE) as model enzyme to test the computational directed evolution protocol for its ability to predict experimentally observed target KE mutants in two rounds of directed evolution experiments. KE

is selected for three reasons. First, the crystal structure of KE has been determined, which provides an accurate template to generate structural models for mutants. Second, the lead variants associated with each round of KE screening in directed evolution have been reported.[31] This provides reference data to examine the predictive capability of computational protocol. Third, KE is known to involve a general acid-base mechanism – the substrate 5-nitrobenzisoxazole is deprotonated by a nearby carboxylate (side chain of Glu or Asp) to form 2-hydroxy-5-nitrobenzonitrile via one single transition state.[30] The relatively simple mechanistic picture reduces the complexity of having to consider the variation of rate-limiting step upon mutation.

We selected two out of seven continuous rounds of directed evolution (i.e., round 5 and 6) that leads to the identification of KE07[30]. Both rounds use error-prone PCR for mutagenesis. Experimentally, round 5 starts from six KE variants and conducts error-prone PCR with 1±1 mutations per mutant. Twelve active variants (only two were reported, Figure 5) from round 5 were filtered into round 6 for error-prone PCR with 3±1 mutations per mutant. This results in twenty active variants (only two were reported, Figure 5) that go into round 7.

Based on the experimental protocol, we set up a computational workflow of directed evolution, aiming to test whether we are able to identify the experimentally-observed target variants from a bunch of randomly-generated mutations through the *in silico* screening protocol (Figure 5). Specifically, for round 5, the workflow starts from six variants, forms a library of sixty mutants with single amino acid substitution through generating ten random mutants for each starting variant, computes thermostability score and electrostatic stabilization energy for each mutant by MD and QM simulations, and selects twelve top-ranked mutants with stable fold and strong electric field to promote deprotonation in the rate-limiting transition state (Figure 5). These twelve mutants (same number as the experimental setup) are used as starting variants to initiate

the round 6. For round 6, random triple mutations are generated with ten mutants for each starting variant to form a library of 120 mutants. This round of screening results in twenty top-ranked mutants (same number as the experimental setup). For each round, we added the two experimentally-observed target variants in the mutant library (i.e., 62 and 122 mutants in total for round 5 and 6, respectively) and test whether these two mutants can be filtered in as the top-ranked mutants based on the computational scoring of protein thermostability and electrostatic stabilization energy (i.e., ΔG_{elec}). By statistics, the chance of observing the target variants among a group of randomly selected mutants is 3.5% for round 5 and 2.5% for round 6 (Supporting Information, Text S5), which presents a challenging test to the predictive capability of the computational directed evolution protocol.

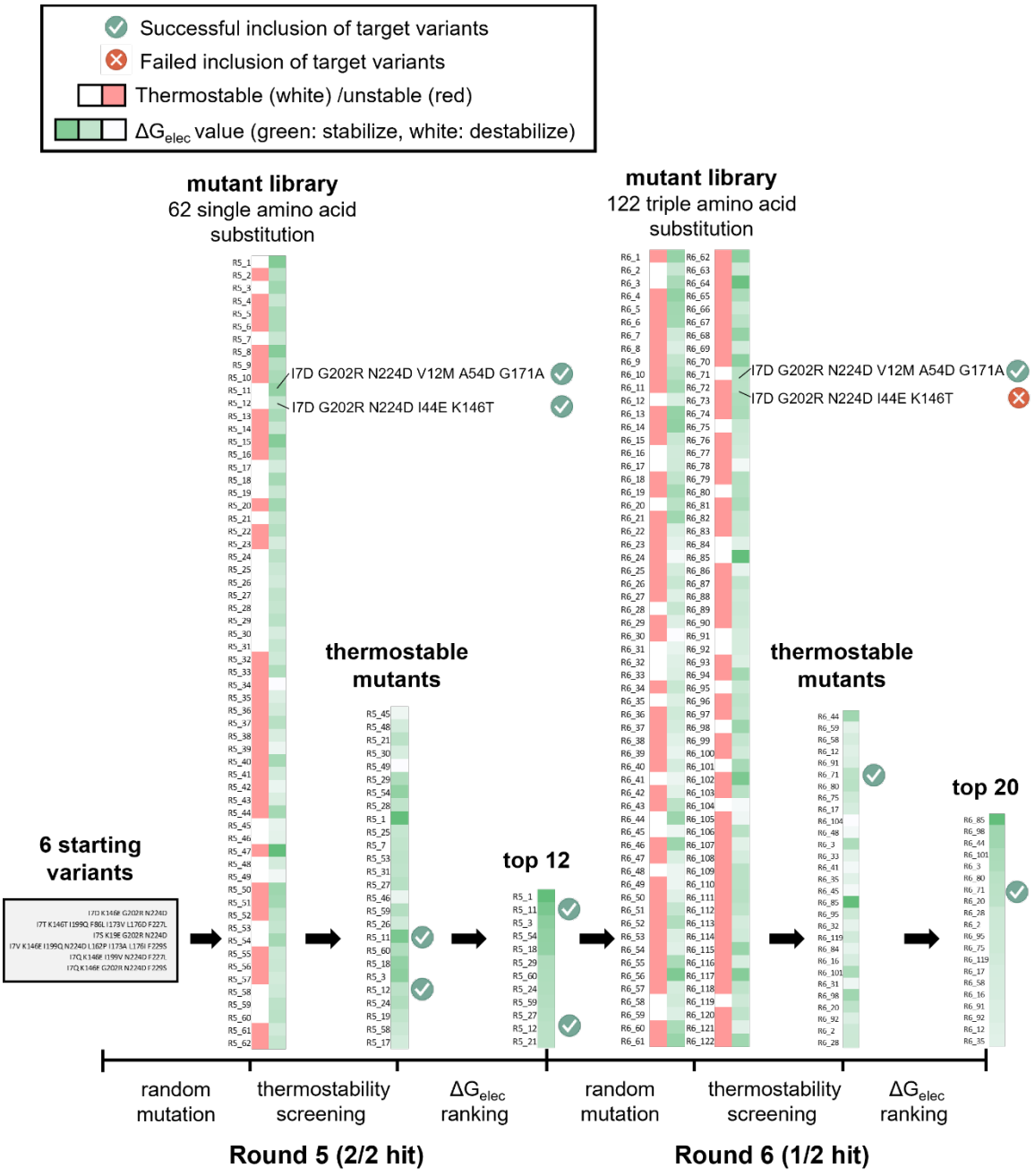

Figure 5. The computational protocol for two rounds of directed evolution. Round 5 and 6 out of the seven rounds of experimental directed evolution of Kemp eliminase are used in the test. Specific stages of the workflow are shown on the bottom axis from left to right. The results of each stage are listed above the axis. Specific information about each mutant is listed in the Supporting Information, Table S1. The modeling results of thermostability and electrostatic stabilization

energy (i.e., ΔG_{elec}) are represented as color-coded blocks next to the mutants. For thermostability score, the thermally stable and unstable mutants are shown in white and red, respectively. For ΔG_{elec}, a color gradient scheme from darker green to white is applied, where darker green refers to strong electric field with greater capability of stabilizing the bond cleavage. "Green check" or "red cross" represents the successful or failed inclusion of an experimentally-observed target mutant in a screening stage.

Figure 5 shows the outcome of each screening stage of the computational directed evolution protocol. In the mutant library of round 5 (i.e., 62 mutants), the thermostability scores vary from -6.4 to 56.3 Rosetta energy unit (REU) with a standard deviation of 17.3 REU; 16 mutants are predicted to be more stable than the wild-type (i.e., negative thermostability score) and 46 less stable (i.e., positive thermostability score). For screening, we defined the "stable" mutants (labeled in white, Figure 5) to consist of all mutants that are more stable than the wild-type (i.e., 16) plus 20% of the top-ranked mutants with a positive thermostability score (i.e., 10). We allowed for a slight magnitude of stability dropping because the loss of stability might be a potential trade-off for catalytic enhancement. Notably, the empirical cutoff value used here was defined prior to the computation and used consistently for both round 5 and round 6. Among the 26 stable mutants, the ΔG_{elec} values vary from 0.5 to 3.1 kcal/mol with a standard deviation of 0.6 kcal/mol; the 12 mutants with top-ranked ΔG_{elec} values were sent for screening in round 6. Throughout the round 5, both experimentally-observed target mutants are found in the final list.

In the mutant library of round 6 (i.e., 122 mutants), the thermostability scores vary from -12.5 REU to 68.3 REU with a standard deviation of 13.9 REU. Using the same screening strategy as described for round 5, the workflow identifies 34 stable mutants. Unfortunately, one of the target mutants (i.e., I7D G202R N224D I44E K146T) is excluded from the list. This is likely caused by

limited accuracy of Rosetta cartesian ddG in predicting multiple mutations that involve epistatic effects. Using data-driven models that account for cooperative contributions of multiple mutations can potentially fix the problem. Among the stable mutants, ΔG_{elec} values vary from -0.4 kcal/mol to 2.5 kcal/mol with a standard deviation of 0.6 kcal/mol. Based on the ranking of ΔG_{elec} values, we selected 20 mutants, in which the other target mutant is included.

Overall, the computational directed evolution protocol identifies three out of four experimentally-observed target mutations in two continuous rounds of mutagenesis. The hit rate is significantly higher than that derived from random sampling. Accelerated by ARMer, the entire workflow finishes in 3 days with ~30 GPUs and ~1,000 CPUs. This corresponds to a total of 18.4 μs MD and 18,400 QM single point calculations. For the next step of development, the challenge lies in the development of catalytically relevant descriptors and predictive scoring functions to inform the impact of mutation on enzyme catalysis. How to incorporate calculations of reaction barriers in the high-throughput workflow is also a meaningful question.

4. Conclusion

We designed a computational directed evolution protocol based on EnzyHTP. Through iterative rounds of screening, the workflow can generate random mutations and identify mutants with enhanced catalytic functions. To accelerate the throughput capability, we developed a Python library, ARMer, to adaptively allocate resources for computational sub-tasks in a high-throughput molecular modeling workflow in HPC clusters. The ARMer Python library consists of two classes: the Job class that defines variables and functions for job configuration, submission, and monitoring; the HPC class that supports the Job class to mediate external input/output in HPC clusters and users can easily customize a subclass for their local HPC clusters. Using commands

implemented in ARMer, a "workflow script" can run on a single CPU thread to generate, submit, and monitor new jobs that call external software to execute sub-tasks.

To test the efficiency, we employed ARMer to manage the sub-tasks in the enzyme modeling workflow for FAcD using EnzyHTP. The sub-tasks include enzyme mutant structure construction, 100 ns MD simulation, 100 QM calculations, and electronic structure analysis. We compared the consumption of time and resource between two allocation strategies: fixed resource allocation versus adaptive resource allocation. Fixed resource allocation involves significant resource waste: in the 14 hours of MD simulation, only one GPU is used but the CPUs are primarily in idle mode (time-waste: 8x14 = 112 CPU hours); in the 3 hours of QM calculations, CPUs are used but the GPU is not (time-waste: 3 GPU hours). In contrast, the adaptive resource allocation makes full use of both computing resources in the corresponding sub-tasks of the workflow. Moreover, ARMer allows parallel submission of multiple smaller computational jobs in a job array and provides customized environment settings for each software used in the workflow.

Enabled by ARMer and EnzyHTP, we constructed the computational directed evolution protocol and tested it against two continuous rounds of experimental directed evolution for Kemp elimiase, KE07. Using the workflow, we successfully reproduced three out of four experimentally-observed target mutants in 3 days with a total of 18.4 μs MD and 18,400 QM single point calculations by ~30 GPUs and ~1000 CPUs. The computational directed evolution protocol holds great promise to accurate the process of enzyme engineering.

ASSOCIATED CONTENT

Supporting Information. The architecture of the ARMer Python library; Example code of ARMer API; Structure of FAcD K83D mutant; Structure of the active site cluster; The comparison

of computing resource cost; The comparison of wall-clock time cost; Details of the benchmark workflow; Choice of parallel strategy; Discrepancy of expected speed up and actual speed up from ARMer parallelization; List of mutants in the computational directed evolution workflow (PDF)

Input files for the fixed/adaptive resource allocation test: input structure; workflow python script; job submission script of the workflow script. Input files for the computational directed evolution test: input structure; workflow python script; job submission script of the workflow script (ZIP).

Data and Software Availability. The code and sample input for ARMer is publically available at https://github.com/ChemBioHTP/ARMer. The input files and structures are provided as part of the SI files. EnzyHTP is available from https://github.com/ChemBioHTP/EnzyHTP/tree/develop. Workflow scripts for the computational directed evolution protocol are available from https://github.com/ChemBioHTP/EnzyHTP/blob/develop/Test_file/KE_accre/R6/KE-metrics_complete.py. AMBER 19 is available from http://ambermd.org/. Gaussian 16 is available from https://gaussian.com/.

AUTHOR INFORMATION

Corresponding Author

*Email: zhongyue.yang@vanderbilt.edu phone: 615-343-9849

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENT

This research was supported by the startup grant from Vanderbilt University and the fellowship of Vanderbilt Institute of Chemical Biology. The PI thanks the sponsorship from Rosetta Commons

Seed Grant Award. This work was also supported by the National Institute of General Medical Sciences of the National Institutes of Health under award number R35GM146982. This work used SDSC Dell Cluster with AMD Rome HDR IB at Expanse from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants BIO200057.[32]

References

1. Arnold, F. H.; Volkov, A. A., Directed evolution of biocatalysts. *Curr. Opin. Chem. Biol.* **1999**, 3, 54-59.
2. Wang, Y.; Xue, P.; Cao, M.; Yu, T.; Lane, S. T.; Zhao, H., Directed Evolution: Methodologies and Applications. *Chem. Rev.* **2021**.
3. Jones, K. A.; Snodgrass, H. M.; Belsare, K.; Dickinson, B. C.; Lewis, J. C., Phage-Assisted Continuous Evolution and Selection of Enzymes for Chemical Synthesis. *ACS Central Science* **2021**, 7, 1581-1590.
4. Tishkov, V. I.; Pometun, A. A.; Stepashkina, A. V.; Fedorchuk, V. V.; Zarubina, S. A.; Kargov, I. S.; Atroshenko, D. L.; Parshin, P. D.; Shelomov, M. D.; Kovalevski, R. P.; Boiko, K. M.; Eldarov, M. A.; D'Oronzo, E.; Facheris, S.; Secundo, F.; Savin, S. S., Rational Design of Practically Important Enzymes. *Moscow Univ. Chem. Bull. (Engl. Transl.)* **2018**, 73, 1-6.
5. Gao, L.; Zou, Y.; Liu, X.; Yang, J.; Du, X.; Wang, J.; Yu, X.; Fan, J.; Jiang, M.; Li, Y.; Houk, K. N.; Lei, X., Enzymatic control of endo- and exo-stereoselective Diels–Alder reactions with broad substrate scope. *Nature Catalysis* **2021**, 4, 1059-1069.
6. Narayan, A. R. H.; Jiménez-Osés, G.; Liu, P.; Negretti, S.; Zhao, W.; Gilbert, M. M.; Ramabhadran, R. O.; Yang, Y.-F.; Furan, L. R.; Li, Z.; Podust, L. M.; Montgomery, J.; Houk, K. N.; Sherman, D. H., Enzymatic hydroxylation of an unactivated methylene C–H bond guided by molecular dynamics simulations. *Nature Chemistry* **2015**, 7, 653-660.
7. Vavra, O.; Damborsky, J.; Bednar, D., Fast approximative methods for study of ligand transport and rational design of improved enzymes for biotechnologies. *Biotechnol. Adv.* **2022**, 60, 108009.
8. Chica, R. A.; Doucet, N.; Pelletier, J. N., Semi-rational approaches to engineering enzyme activity: combining the benefits of directed evolution and rational design. *Curr. Opin. Biotechnol.* **2005**, 16, 378-384.
9. Bhowmick, A.; Sharma, S. C.; Honma, H.; Head-Gordon, T., The role of side chain entropy and mutual information for improving the de novo design of Kemp eliminases KE07 and KE70. *Physical Chemistry Chemical Physics* **2016**, 18, 19386-19396.
10. Khersonsky, O.; Lipsh, R.; Avizemer, Z.; Ashani, Y.; Goldsmith, M.; Leader, H.; Dym, O.; Rogotner, S.; Trudeau, D. L.; Prilusky, J.; Amengual-Rigo, P.; Guallar, V.; Tawfik, D. S.; Fleishman, S. J., Automated Design of Efficient and Functionally Diverse Enzyme Repertoires. *Mol. Cell* **2018**, 72, 178-186.e5.
11. Romero-Rivera, A.; Garcia-Borràs, M.; Osuna, S., Role of Conformational Dynamics in the Evolution of Retro-Aldolase Activity. *ACS Catalysis* **2017**, 7, 8524-8532.

12. Fried, S. D.; Boxer, S. G., Electric Fields and Enzyme Catalysis. *Annu. Rev. Biochem* **2017**, 86, 387-415.
13. Jiang, Y.; Yan, B.; Chen, Y.; Juarez, R. J.; Yang, Z. J., Molecular Dynamics-Derived Descriptor Informs the Impact of Mutation on the Catalytic Turnover Number in Lactonase Across Substrates. *The Journal of Physical Chemistry B* **2022**, 126, 2486-2495.
14. Clements, H.; Flynn, A.; Nicholls, B.; Grosheva, D.; Hyster, T.; Sigman, M., In; Chemrxiv: 2021.
15. Xie, W. J.; Warshel, A., Natural Evolution Provides Strong Hints about Laboratory Evolution of Designer Enzymes. *Proc. Natl. Acad. Sci. U.S.A.* **2022**, 119, e2207904119.
16. Jiang, Y.; Ran, X.; Yang, Z. J., Data-Driven Enzyme Engineering to Identify Function-Enhancing Enzymes. *Protein Engineering, Design and Selection* **2022**.
17. Shao, Q.; Jiang, Y.; Yang, Z. J., EnzyHTP: A High-Throughput Computational Platform for Enzyme Modeling. *J. Chem. Inf. Model.* **2022**, 62, 647-655.
18. Amrein, B. A.; Steffen-Munsberg, F.; Szeler, I.; Purg, M.; Kulkarni, Y.; Kamerlin, S. C. L., CADEE: Computer-Aided Directed Evolution of Enzymes. *IUCrJ* **2017**, 4, 50-64.
19. Makinen, M. W.; Fink, A. L., Reactivity and Cryoenzymology of Enzymes in the Crystalline State. *Annual Review of Biophysics and Bioengineering* **1977**, 6, 301-343.
20. Schulz, E. C.; Mehrabi, P.; Müller-Werkmeister, H. M.; Tellkamp, F.; Jha, A.; Stuart, W.; Persch, E.; De Gasparo, R.; Diederich, F.; Pai, E. F.; Miller, R. J. D., The hit-and-return system enables efficient time-resolved serial synchrotron crystallography. *Nat. Methods* **2018**, 15, 901-904.
21. Mehrabi, P.; Di Pietrantonio, C.; Kim, T. H.; Sljoka, A.; Taverner, K.; Ing, C.; Kruglyak, N.; Pomès, R.; Pai, E. F.; Prosser, R. S., Substrate-Based Allosteric Regulation of a Homodimeric Enzyme. *J. Am. Chem. Soc.* **2019**, 141, 11540-11556.
22. Kim, T. H.; Mehrabi, P.; Ren, Z.; Sljoka, A.; Ing, C.; Bezginov, A.; Ye, L.; Pomès, R.; Prosser, R. S.; Pai, E. F., The role of dimer asymmetry and protomer dynamics in enzyme catalysis. *Science* **2017**, 355, eaag2355.
23. Mehrabi, P.; Schulz, E. C.; Dsouza, R.; Müller-Werkmeister, H. M.; Tellkamp, F.; Miller, R. J. D.; Pai, E. F., Time-resolved crystallography reveals allosteric communication aligned with molecular breathing. *Science* **2019**, 365, 1167-1170.
24. Park, H.; Bradley, P.; Greisen, P.; Liu, Y.; Mulligan, V. K.; Kim, D. E.; Baker, D.; Dimaio, F., Simultaneous Optimization of Biomolecular Energy Functions on Features from Small Molecules and Macromolecules. *J. Chem. Theory Comput.* **2016**, 12, 6201-6212.
25. Frenz, B.; Lewis, S. M.; King, I.; DiMaio, F.; Park, H.; Song, Y., Prediction of Protein Mutational Free Energy: Benchmark and Sampling Improvements Increase Classification Accuracy. *Frontiers in Bioengineering and Biotechnology* **2020**, 8.
26. Fried, S. D.; Boxer, S. G., Measuring Electric Fields and Noncovalent Interactions Using the Vibrational Stark Effect. *Acc. Chem. Res.* **2015**, 48, 998-1006.
27. Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Scalmani, G.; Barone, V.; Petersson, G. A.; Nakatsuji, H.; Li, X.; Caricato, M.; Marenich, A. V.; Bloino, J.; Janesko, B. G.; Gomperts, R.; Mennucci, B.; Hratchian, H. P.; Ortiz, J. V.; Izmaylov, A. F.; Sonnenberg, J. L.; Williams; Ding, F.; Lipparini, F.; Egidi, F.; Goings, J.; Peng, B.; Petrone, A.; Henderson, T.; Ranasinghe, D.; Zakrzewski, V. G.; Gao, J.; Rega, N.; Zheng, G.; Liang, W.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Vreven, T.; Throssell, K.; Montgomery Jr., J. A.; Peralta, J. E.; Ogliaro, F.; Bearpark, M. J.; Heyd, J. J.; Brothers, E. N.; Kudin, K. N.; Staroverov, V. N.;

Keith, T. A.; Kobayashi, R.; Normand, J.; Raghavachari, K.; Rendell, A. P.; Burant, J. C.; Iyengar, S. S.; Tomasi, J.; Cossi, M.; Millam, J. M.; Klene, M.; Adamo, C.; Cammi, R.; Ochterski, J. W.; Martin, R. L.; Morokuma, K.; Farkas, O.; Foresman, J. B.; Fox, D. J. *Gaussian 16 Rev. C.01*, Wallingford, CT, 2016.
28. Lu, T.; Chen, F., Multiwfn: A multifunctional wavefunction analyzer. *J. Comput. Chem.* **2012**, 33, 580-592.
29. D.A. Case, H. M. A., K. Belfon, I.Y. Ben-Shalom, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, III, G.A. Cisneros, V.W.D. Cruzeiro, T.A. Darden, R.E. Duke, G. Giambasu, M.K. Gilson, H. Gohlke, A.W. Goetz, R. Harris, S. Izadi, S.A. Izmailov, C. Jin, K. Kasavajhala, M.C. Kaymak, E. King, A. Kovalenko, T. Kurtzman, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, M. Machado, V. Man, M. Manathunga, K.M. Merz, Y. Miao, O. Mikhailovskii, G. Monard, H. Nguyen, K.A. O'Hearn, A. Onufriev, F. Pan, S. Pantano, R. Qi, A. Rahnamoun, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C.L. Simmerling, N.R. Skrynnikov, J. Smith, J. Swails, R.C. Walker, J. Wang, H. Wei, R.M. Wolf, X. Wu, Y. Xue, D.M. York, S. Zhao, and P.A. Kollman Amber 2021.
30. Röthlisberger, D.; Khersonsky, O.; Wollacott, A. M.; Jiang, L.; Dechancie, J.; Betker, J.; Gallaher, J. L.; Althoff, E. A.; Zanghellini, A.; Dym, O.; Albeck, S.; Houk, K. N.; Tawfik, D. S.; Baker, D., Kemp elimination catalysts by computational enzyme design. *Nature* **2008**, 453, 190-195.
31. Khersonsky, O.; Röthlisberger, D.; Dym, O.; Albeck, S.; Jackson, C. J.; Baker, D.; Tawfik, D. S., Evolutionary Optimization of Computationally Designed Enzymes: Kemp Eliminases of the KE07 Series. *J. Mol. Biol.* **2010**, 396, 1025-1042.
32. Towns, J.; Cockerill, T.; Dahan, M.; Foster, I.; Gaither, K.; Grimshaw, A.; Hazlewood, V.; Lathrop, S.; Lifka, D.; Peterson, G. D.; Roskies, R.; Scott, J. R.; Wilkins-Diehr, N., XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* **2014**, 16, 62-74.

Table of Contents Graphic