# AlphaFold2 Workflow Optimization for High Throughput Predictions in HPC Environment

Edwin F. Posada
Francisco McGee
Richard Berger
edwin.posada@temple.edu
francisco.mcgee@temple.edu
richard.berger@outlook.com
Institute for Computational Molecular Science, Temple University
Philadelphia, PA, USA

Mitchell Dorrell
Pittsburgh Supercomputing Center
Pittsburgh, PA, USA

Jason Lamanna
jason.lamanna@temple.edu
Institute for Computational Molecular Science, Temple University
Philadelphia, PA, USA

Sergiu Sanielevici
sergiu@psc.edu
Pittsburgh Supercomputing Center
Pittsburgh, PA, USA

Vincenzo Carnevale
vincenzo.carnevale@temple.edu
Institute for Computational Molecular Science, Temple University
Philadelphia, PA, USA

## ABSTRACT

In this work, we propose a high-throughput implementation that executes AlphaFold2 efficiently in a High-Performance Computing environment. In this case, we have tested our proposed workflow with the T1050 CASP14 sequence on PSC's Bridges-2 HPC system. The results showed an improvement in computation-only runtimes and the opportunity to reuse the protein databases when calculating many structures simultaneously, which would lead to massive time savings while maximizing the utilization of computing resources.

## CCS CONCEPTS

• **Applied computing** → **Physics**; **Bioinformatics**; • **General and reference** → **Performance**.

## KEYWORDS

Software Optimization, AlphaFold2, bioinformatics, High-Performance Computing

**ACM Reference Format:**
Edwin F. Posada, Francisco McGee, Richard Berger, Mitchell Dorrell, Jason Lamanna, Sergiu Sanielevici, and Vincenzo Carnevale. 2022. AlphaFold2 Workflow Optimization for High Throughput Predictions in HPC Environment. In *Practice and Experience in Advanced Research Computing (PEARC '22), July 10–14, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3491418.3535181


*PEARC '22, July 10–14, 2022, Boston, MA, USA*

ACM ISBN 978-1-4503-9161-0/22/07…$15.00
https://doi.org/10.1145/3491418.3535181

## 1 INTRODUCTION

Though separated in some cases by over 1B years of evolutionary time, divergent protein families may share remarkable sequence and structural patterns. Although the patterns are complex, and although the evolutionary parameters that generated them are ambiguous, the patterns are detectable by sophisticated statistical models, given sufficient protein data [1, 6, 12]. A model trained on sufficient data could, in principle, extract these evolutionary parameters from the patterns and then parameterize itself to generate viable synthetic proteins that are statistically indistinguishable from those generated by natural evolutionary processes, but in a controllable way.

Until recently, sufficient volumes of protein structure data were unavailable, so prior generative protein modeling methods focused primarily on sequence data. However, the recent release of the artificial intelligence (AI) model AlphaFold2 (AF2) by Google's DeepMind, which can predict structures with atomic precision on par with experimental techniques, opens the door to a variety of new generative models that would benefit from training on protein structure in addition to the sequence [8, 19]. In 2020, Google's Deepmind entered the 14$^{th}$ CASP competition (CASP14 [9]) with "AlphaFold2" (AF2)[8], whose performance shocked the world of structural bioinformatics, reaching a level of atomic precision which until then was the prerogative of notoriously difficult experimental techniques. The Critical Assessment of Structure Prediction (CASP) is a decades-old international competition for computationally determining protein structures. Subsequently, the CASP14 organizers announced in a press release that AF2 had effectively solved the 50-year-old problem of single-chain protein folding [10], an achievement they said would revolutionize medicine in our lifetime [19].

Until the recent arrival of AF2, the data scarcity issue was even more acute for protein structures than for sequences because structures are notoriously difficult to determine experimentally, and computational methods were nowhere near as accurate. For generative protein sequence models, it has been shown that between

**Table 1: Databases needed for each phase in the proposed workflow. A graphical representation of the proposed workflow can be found in Figure 1.**

| Phase | Process | Database | Size | SquashFS |
|---|---|---|---|---|
| P0 | **jackhmmer** | UniRef90 | 58 GB | 31 GB |
| | | MGnify | 64 GB | 33 GB |
| P0 | **hhblits** | BFD | 1.7 TB | 272 GB |
| | | Uniclust30 | 86 GB | 26 GB |
| P1 | **hhsearch** and Features | PDB70 | 56 GB | 20 GB |
| | | PDB MMCIF | 206 GB | 51 GB |
| P2 | Models | Params | 3.5 GB | 3.3 GB |

$10^4$ to $10^6$ training samples are needed depending on the model and the protein(s) being studied [6, 12]. In contrast, most individual protein families have less than $10^4$ complete sequences available in public databases [6, 12, 14], and even fewer structures. Because AF2 can generate the structure for any given sequence, AF2 has created the opportunity for novel generative protein models that include structure in the training data.

Building such a large structural training sample database implies a substantial computational cost for which workflow/code optimization becomes necessary, specifically on shared computational resources like on-premises High-Performance Computing (HPC) facilities and national supercomputing centers. In the case of AF2, some efforts have been made to run it more efficiently in an HPC setting, such as ParaFold [20] and FastFold [4]. In this short paper, we would like to describe the experience of using AF2 in our main HPC facility at Temple University, also known as Owl's Nest 2, and on Bridges-2 [2] at the Pittsburgh Supercomputing Center (PSC).

This paper is organized as follows: in Section 2, we summarize the AF2 workflow and report on-premises benchmarking for the particular case of the CASP14 T1050 sequence. In Section 3, we present our own high-throughput implementation that runs AF2 more efficiently in an HPC environment. Lastly, we close with some conclusions and future work.

## 2 AF2 PIPELINE AND INITIAL BENCHMARK

The original AF2 workflow is composed of two main parts, labeled P1 and P2 in Fig. 1, left: the Multiple Sequence Alignment (MSA) and Feature Generation (Fig. 1, left, blue), and the Structure Inference and Relaxation (Fig. 1, left, orange). P1 is executed exclusively on the CPU, while the P2 can leverage GPU acceleration.

In P1, AF2 takes as input a protein sequence to generate the MSA by querying a large set of protein sequences from public databases. AF2 uses **jackhmmer** [5] and **hhblits** [17] to search over those databases in two stages. The first stage creates initial MSAs by using **jackhmmer** over UniRef90 [18] (jackhmmer_1) and MGnify [15] (jackhmmer_2), and **hhblits** over BFD + Uniclust30 [13], which then are used in the second stage when the **hhsearch** [17] takes place over PDB70 [17]. Due to the size of the protein databases (Table 1), and the elevated random file access, the MSA creation is the most computationally and time-intensive step. For the T1050 CASP14 sequence, P1 takes between 3.5 to 4 hours to complete (Fig. 2).

In P2, AF2 takes as input the protein sequence, the MSA, and other features from P1. Then, it infers a structure, followed by protein structure relaxation using the AMBER forcefield [3]. By default,

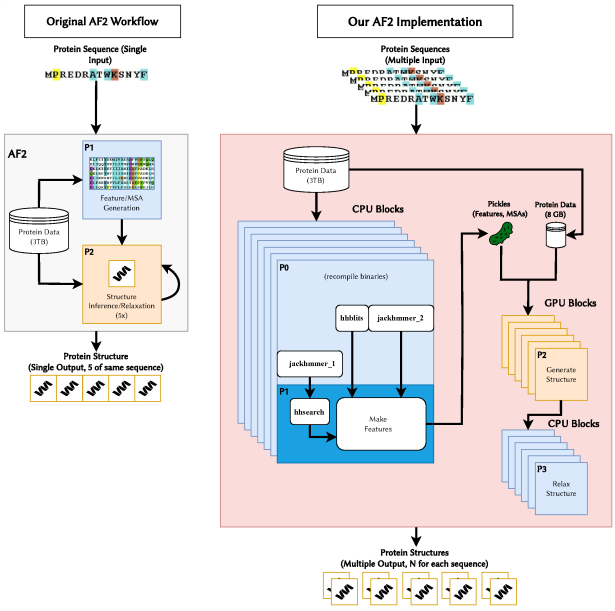


**Figure 1: Left: Original AF2 workflow. One sequence is fed to AF2. In P1, features and MSA are created, then fed to P2, where a structure is inferred, then relaxed using the AMBER [3] forcefield. P2 is repeated 5x to create a total of 5 structures for the same initial protein sequence. All code blocks in P1 and P2 are run sequentially. Right: Our modified AF2 workflow. The input remains the same as in the original, but multiple sequences can now be fed into AF2 asynchronously in parallel. Firstly, we recompiled the binaries for significant runtime savings overall. P1 from the original has been decoupled from P2, then decomposed into P0 and P1. P0 is then further decomposed, and the processes decoupled. P2 from the original has been decomposed into P2 (Structure Generation) and P3 (structure relaxation). Instead of generating five structures as in the original, we generate only one structure from P2 for 80% runtime savings in P2.**

P2 is repeated five times, creating five independent structures for each single sequence submitted to P1. As shown in Fig. 2, the structure inference and relaxation are about 20% of the total execution time for the T1050 sequence, which is 4.8 hours.

## 3 AF2 HIGH-THROUGHPUT WORKFLOW OPTIMIZATION

In order to achieve maximum utilization of computational resources for high-throughput structure prediction with AF2, we employed a general strategy of decomposing and decoupling processes within and across the original phases, P1 and P2. We began by decoupling P1 and P2. The motivation for this is the usual setup in an HPC facility, i.e., it is common to have independent queues for GPU nodes and regular CPU compute nodes. We have also decoupled the inference and relaxation steps in P2 since the inference is the only step that can be executed on GPUs. The proposed workflow is

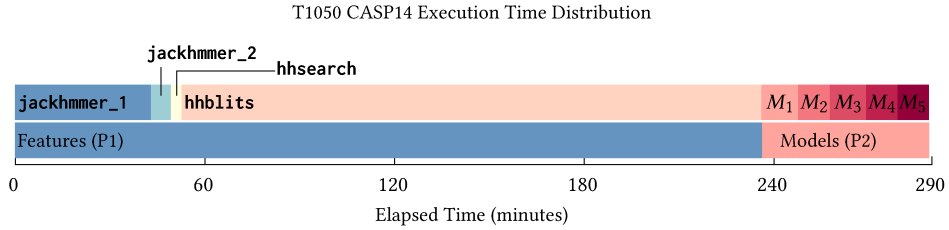


**Figure 2: T1050 CASP14 Execution time distribution. Most of the time is spent in the MSA creation. Calculation performed on Bridges-2 [2].**

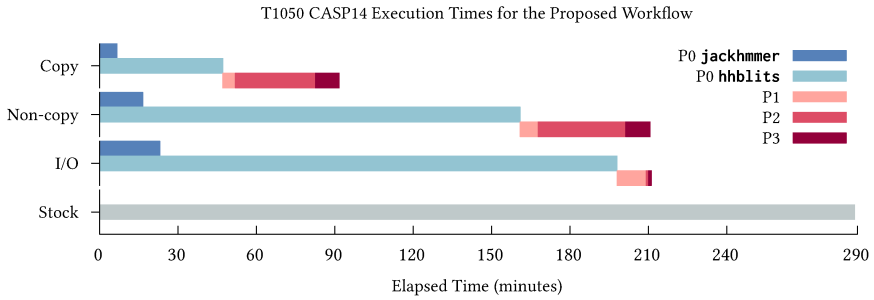


**Figure 3: Execution times for the proposed workflow after compressing databases and phase decoupling. The non-copy version uses the network file system (NFS) to read the databases from a remote location, while the copy strategy uses the databases locally on each compute node after database transfer. Not all databases are transferred; see Table 1 for a complete list of the databases used for each phase of the workflow. P0, P1 and P3 were performed on Bridges-2 Regular Memory (RM) compute nodes, while P2 was calculated on Bridges-2 GPU nodes.**

shown on the right-hand side of Figure 1; P1 and P3 are executed on CPU nodes, while P2 is executed on GPUs.

As mentioned in the previous section, the most time-consuming step from the original AF2 implementation was the MSA creation due to the high network I/O performed over various large protein databases. We present our high-throughput AF2 implementation in two versions: one that is memory-intensive, and one that is network I/O-intensive, referred to as "copy" and "non-copy", respectively 3. First, we describe the memory-intensive version in detail. By carefully decomposing the individual MSA generation steps, we were able to isolate the specific resources used by each step, and exploited this information to decouple the steps from each other, allowing us to send each process to its own node with only the database(s) needed for that step. By transferring the databases directly into memory on the node, network I/O was mitigated. However, we note that the memory-intensive option requires >2TB RAM compute nodes, and would only be beneficial for high-throughput use cases in which many MSAs are calculated in the same job script. With respect to decomposition of the original MSA generation, we observed that **jackhmmer** and **hhblits** could be run independently of each other. As mentioned, running **jackhmmer** and **hhblits** on different compute nodes obviated the need to copy all databases to each node, only those needed for each process running on that node. As a result of the decomposition within the original P1, the initial MSA creation with **jackhmmer** and **hhblits** are denoted as P0 in our optimized implementation, and the subsequent **hhsearch** and feature generation as P1. The list of databases needed for each phase of the workflow (PX) is exhibited in Table 1. These protein databases are easily compressible, and in this work we have used SquashFS [11] to do so.

After SquashFS database compression, we calculated the T1050 CASP14 sequence using our two different strategies. To reiterate, for the memory-intensive "copy" version, we transferred the necessary SquashFS files directly into memory on the compute node where the calculation would take place, and then performed the benchmarking calculations. In the second "non-copy" version, we did not copy the databases, but rather queried them using an IPoIB (IP-over-InfiniBand) network file system. The times for each strategy are listed in Table 3.

The "non-copy" version takes 3.5 hours compared with the 4.8 hours used by the "stock" version of AF2. The runtime improvement is even more noticeable with the "copy" version of the workflow, just 1.5 hours of computation-only. It is important to note that if we add the network I/O time; i.e., the time it takes to transfer the data over the network; plus decompression of the databases and the computation time, the overall runtime is 5 hours. However, moving the databases to the local disk is a one-time cost when running multiple calculations, which is expected in a real-world high-throughput use case.

## 4 CONCLUSIONS AND FUTURE WORK

Any modern supercomputing center would struggle to produce enough structural data from the default AF2 installation to feed even a modest generative protein model. This goes to show that

having access to the biggest hammer in the workshop is pointless if it is too heavy for anyone to lift. It is on this point that our work here gains traction. If made available to the public on PSC's systems, which is our current goal, our high-throughput AF2 implementation could handle the undifferentiated heavy lifting of HPC, allowing researchers in generative protein modeling to wield one of the most potent and promising tools for scientific discovery in our lifetime.

The original AF2 implementation calls out to several external utilities as part of an integrated data-processing pipeline, specifically **jackhmmer**, **hhblits**, **hhsearch**, **hmmsearch**, **hmmbuild**, and **kalign**, although not all of these utilities are needed for every execution. Each of these utilities is a standalone tool, with its own performance tuning options. In an effort to provide a general-purpose pipeline, the original AF2 implementation makes conservative tuning choices to ensure reasonable performance across the commonly available hardware. However, laptops and workstations have very different hardware characteristics from nodes of a high performance computing cluster, which can benefit from much greater levels of parallelization and which may offer hardware acceleration. To achieve greater flexibility across a wide variety of hardware, the AF2 pipeline can be further disassembled. The utilities can be run in a standalone manner, such that the runtime options of each utility can be chosen by the user to optimize the performance on the actual hardware present. Similar to the decoupling strategies discussed above, the user is free to run the utilities concurrently as well.

By default, the original AF2 implementation will run **jackhmmer** twice, followed by **hhsearch**, then **hhblits**, before finally invoking the AF2 algorithm itself to predict the structure, followed by relaxation using stand techniques. With a fully decoupled pipeline, users perform this same task by submitting independent jobs to the HPC scheduler for each of the above steps, taking advantage of job-dependency features to inform the scheduler that **hhsearch** depends on the output from **jackhmmer**, and that the final structure generation and relaxation depends on the output from all of the prior external-utility jobs. The final benefit to this approach is that alternative software can be chosen to replace certain utilities in order to achieve even greater acceleration. With the data-preparation techniques described above, the current external utilities are still limited by inefficient scaling and computational capacity, and so still dominate the execution time. As refined implementations of these utilities are developed, the extra decoupling described in this paper will enable users to choose which implementation to use with AF2 without having to make any manual changes to the AF2 code base.

In perspective, decoupling and optimizing AF2 as we have done here will enable a number of diverse applications in computational structural biology by leveraging the computational capability of high performance computing facilities. One possible application is *de novo* protein design. In this context, the AF2 protein structure prediction engine could be used as a component of a neural-network based probabilistic generative model; preliminary results obtained in this field by our group are encouraging and suggest that, in its optimized version, AF2 could be used to generate, on demand, tens of thousands of protein structures. Another interesting user scenario could involve interactive molecular visualization systems like VMD [7] or PyMol [16], which are typically used to visually inspect and analyze experimental structures. An appropriately optimized and decoupled AF2 could, for instance, be used to visualize the structure corresponding to a sequence provided by the user within the environment of the molecular visualization system without the need to download or copy additional files.

## ACKNOWLEDGMENTS

This work was funded in part by the National Institutes of Health (R01GM093290, S10OD020095, and R01GM131048; V.C.), and the National Science Foundation through Grant no. IOS-1934848 (V.C.). This research includes calculations carried out on HPC resources supported in part by the National Science Foundation through major research instrumentation grant number 1625061 and through the Extreme Science and Engineering Discovery Environment (XSEDE), grant number ACI-1548562; as well as by the US Army Research Laboratory under contract number W911NF-16-2-0189.

## REFERENCES

[1] Mohammed AlQuraishi. 2019. ProteinNet: a standardized data set for machine learning of protein structure. *BMC Bioinformatics* 20, 1 (June 2019), 311. https://doi.org/10.1186/s12859-019-2932-0

[2] Shawn T Brown, Paola Buitrago, Edward Hanna, Sergiu Sanielevici, Robin Scibek, and Nicholas A Nystrom. 2021. Bridges-2: A Platform for Rapidly-Evolving and Data Intensive Research. In *Practice and Experience in Advanced Research Computing*. 1–4.

[3] David A Case, H Metin Aktulga, Kellon Belfon, Ido Ben-Shalom, Scott R Brozell, David S Cerutti, Thomas E Cheatham III, Vinícius Wilian D Cruzeiro, Tom A Darden, Robert E Duke, et al. 2021. *Amber 2021*. University of California, San Francisco.

[4] Shenggan Cheng, Ruidong Wu, Zhongming Yu, Binrui Li, Xiwen Zhang, Jian Peng, and Yang You. 2022. FastFold: Reducing AlphaFold Training Time from 11 Days to 67 Hours. https://doi.org/10.48550/ARXIV.2203.00854

[5] Sean R Eddy. 2011. Accelerated profile HMM searches. *PLoS computational biology* 7, 10 (2011), e1002195.

[6] Allan Haldane and Ronald M. Levy. 2019. Influence of multiple-sequence-alignment depth on Potts statistical models of protein covariation. *Physical Review. E* 99, 3-1 (March 2019), 032405. https://doi.org/10.1103/PhysRevE.99.032405

[7] William Humphrey, Andrew Dalke, and Klaus Schulten. 1996. VMD: visual molecular dynamics. *Journal of molecular graphics* 14, 1 (1996), 33–38.

[8] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (Aug. 2021), 583–589. https://doi.org/10.1038/s41586-021-03819-2

[9] Andriy Kryshtafovych, Torsten Schwede, Maya Topf, Krzysztof Fidelis, and John Moult. 2021. Critical assessment of methods of protein structure prediction (CASP)—Round XIV. *Proteins: Structure, Function, and Bioinformatics* 89, 12 (2021), 1607–1617.

[10] Cyrus Levinthal. 1969. How to fold graciously. *Mossbauer spectroscopy in biological systems* 67 (1969), 22–24.

[11] Phillip Lougher and R Lougher. 2008. SquashFS.

[12] Francisco McGee, Quentin Novinger, Ronald M. Levy, Vincenzo Carnevale, and Allan Haldane. 2021. Generative Capacity of Probabilistic Protein Sequence Models. arXiv:2012.02296 [cs.LG]

[13] Milot Mirdita, Lars Von Den Driesch, Clovis Galiez, Maria J Martin, Johannes Söding, and Martin Steinegger. 2017. Uniclust databases of clustered and deeply annotated protein sequences and alignments. *Nucleic acids research* 45, D1 (2017), D170–D176.

[14] Jaina Mistry, Sara Chuguransky, Lowri Williams, Matloob Qureshi, Gustavo A Salazar, Erik LL Sonnhammer, Silvio CE Tosatto, Lisanna Paladin, Shriya Raj, Lorna J Richardson, et al. 2021. Pfam: The protein families database in 2021. *Nucleic acids research* 49, D1 (2021), D412–D419.

[15] Alex L Mitchell, Alexandre Almeida, Martin Beracochea, Miguel Boland, Josephine Burgin, Guy Cochrane, Michael R Crusoe, Varsha Kale, Simon C Potter, Lorna J Richardson, et al. 2020. MGnify: the microbiome analysis resource in 2020. *Nucleic acids research* 48, D1 (2020), D570–D578.

[16] Schrödinger, LLC. 2015. The PyMOL Molecular Graphics System, Version 1.8. (November 2015).
[17] Martin Steinegger, Markus Meier, Milot Mirdita, Harald Vöhringer, Stephan J Haunsberger, and Johannes Söding. 2019. HH-suite3 for fast remote homology detection and deep protein annotation. *BMC bioinformatics* 20, 1 (2019), 1–15.
[18] Baris E Suzek, Yuqi Wang, Hongzhan Huang, Peter B McGarvey, Cathy H Wu, and UniProt Consortium. 2015. UniRef clusters: a comprehensive and scalable alternative for improving sequence similarity searches. *Bioinformatics* 31, 6 (2015), 926–932.
[19] Rob Toews. 2021. AlphaFold Is The Most Important Achievement In AI—Ever. https://www.forbes.com/sites/robtoews/2021/10/03/alphafold-is-the-most-important-achievement-in-ai-ever/ Section: AI.
[20] Bozitao Zhong, Xiaoming Su, Minhua Wen, Sichen Zuo, Liang Hong, and James Lin. 2021. ParaFold: Paralleling AlphaFold for Large-Scale Predictions. https://doi.org/10.48550/ARXIV.2111.06340