Understanding the Performance Behaviors of End-to-End Protein Design Pipelines on GPUs

Jinwoo Hwang, Yeongmin Hwang, Tadiwos Meaza, Hyeonbin Bae, *Member, IEEE*
Jongse Park, *Senior Member, IEEE*

***Abstract*—Recent computational advances enable protein design pipelines to run end-to-end on GPUs, yet their heterogeneous computational behaviors remain undercharacterized at the system level. We implement and profile a representative pipeline at both component and full-pipeline granularities across varying inputs and hyperparameters. Our characterization identifies generally low GPU utilization and high sensitivity to sequence length and sampling strategies. We outline future research directions based on these insights and release an open-source pipeline and profiling scripts to facilitate further studies.**

***Index Terms*—Protein design, protein engineering, characterization, bioinformatics**

I. Introduction

PROTEINS are the fundamental molecular machinery driving biological systems. Recent advances in protein AI models [1]–[8] have significantly reduced the cost of computational protein structure prediction, enabling the design of proteins with enhanced or novel functionalities relevant to next-generation therapeutics, industrial biocatalysts, and biosensors [9]. Protein design pipelines typically involve multiple stages with distinct computational behaviors and heavy GPU dependencies, yet their combined system-level performance implications remain largely unexplored.

We address this gap by systematically profiling a representative GPU-based protein design pipeline at both component and pipeline granularities, examining performance sensitivity to sequence length, hyperparameters, and sampling strategies. Our study investigates GPU utilization patterns and the impacts of GPU scheduling under co-location scenarios. We also outline directions for future research, including developing cost-aware metrics, scaling to multi-GPU environments, and exploring agent-assisted orchestration. To foster reproducibility and accelerate future studies, we release our pipeline and profiling tools as open-source software. [1]

II. Background and Motivation

A. Proteins and In Silico Screening

Protein engineering. The diversity of protein function arises from their three-dimensional structures encoded by amino acid sequences. Changing the order and composition of amino acids can therefore alter a protein's function. Historically, discovery and optimization heavily relied on physical screening and structural characterization, which are costly and slow, limiting exploration of the protein design space.

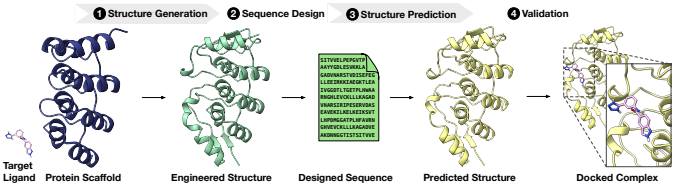

Fig. 1. Four major stages of a protein-design pipeline.

AI-driven in silico screening. Recent advancements in AI have transformed this workflow by providing reliable structural and functional predictions at scale. With these models, the marginal cost of evaluating designs computationally is significantly lower compared to traditional wet-lab experiments. Consequently, design cycles that previously took months now require only hours using modest GPU clusters, paving the way for a new paradigm of rapid, in silico screening.

High-throughput screening. This paradigm involves rapidly evaluating extensive virtual libraries of protein candidates. Early computational filters assess candidates based on functionality, manufacturability, and stability, eliminating unsuitable designs early. The remaining high-priority candidates receive focused experimental attention, advancing only the strongest designs to physical validation. Under such conditions, high-throughput in silico screening becomes critical. Large batches increase the exploration of the design space and enhance the probability of identifying superior candidates [10].

B. Workflow of Protein Design

Protein design workflows vary depending on the specific objectives and are assembled from interchangeable components. Nonetheless, they commonly follow a four-stage pattern, as depicted in Figure 1.

❶ **Structure generation.** Structure generation proposes engineered protein structures satisfying geometric or functional constraints, using models such as RFdiffusion [2]. Structures can be designed from scratch or derived from a known scaffold protein, often specifying target ligands for optimization.

❷ **Sequence design.** Sequence design generates amino acid sequences that match the given structures, using inverse folding models such as ProteinMPNN [3]. Protein language models, including ESM-2 [4], further refine and explore sequences.

❸ **Structure prediction.** Structure prediction folds sequences back into three-dimensional structures with predictors like AlphaFold 3 [1], ESMFold [4], and Protenix [6]. Most models other than ESMFold require evolutionary context, performing homology searches and sequence alignments via MMseqs2 [5].

❹ **Validation.** Evaluation estimates key properties, such as binding affinity and structural stability, by forming docked

Manuscript received 13 November 2025; accepted 16 December 2025. Date of publication 19 December 2025; date of current version 6 January 2026. Digital Object Identifier 10.1109/LCA.2025.3646250. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2024-00342148).

The authors are with the School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, 34141, South Korea. (e-mail: jwhwang,ymhwang,tadiwos,hbbae,jspark@casys.kaist.ac.kr).

[1]https://github.com/casys-kaist/protein-design-pipelines.git

TABLE I
SELECTED SCAFFOLDS AND LIGANDS. WE REPORT PDB IDS WITH LENGTHS: AMINO ACID RESIDUES (AA) AND HEAVY ATOMS (HA).

	Short	Medium	Long
Scaffold ID (AA)	1SHG (62)	1N0R (126)	1TIM (247)
Ligand ID (HA)	3DX1 (9)	4DE1 (23)	3PRS (50)

complexes and applying scoring methods like AutoDock Vina and DiffDock [7], [11], [12]. For comprehensive context and tool comparisons, refer to a recent survey [13].

C. Heterogeneity across Components

Diverse workloads. Components employ varied model architectures and kernels, ranging from Diffusion [1], [2], [6], [7], Transformers [1], [4], [6], graph neural networks [3], custom GPU kernels for sequence alignment [5], and scoring methodologies [12]. This architectural diversity results in distinct computational patterns, activation footprints, and kernel behaviors. Inputs and hyperparameters further amplify this diversity. Some components are sensitive to sequence length, and several expose adjustable hyperparameters balancing throughput and accuracy, such as diffusion step counts.

Need for characterization. These insights motivate two complementary analyses. Initially, we characterize individual components across various inputs and hyperparameters. Subsequently, we evaluate the complete end-to-end pipeline performance on shared GPUs, analyzing scheduling interactions and performance implications at the system level.

III. PROTEIN DESIGN PIPELINE

A. Protein Engineering Scenario

We illustrate a protein engineering scenario wherein a stable, well-characterized scaffold protein is modified to engage a specific small-molecule target ligand.

Protein and ligand pairs. We select four protein scaffolds and three ligands to represent a diverse range of sequence lengths and complexities, as detailed in Table I. For protein scaffolds, we use well-established structures spanning short to long sequences. For ligands, we sample three molecules from the CASF-2016 [14] dataset, choosing examples at the 1st, 50th, and 99th percentiles by length.

Components and hyperparameters. We select widely adopted models for each pipeline component to reflect current industry standards, as outlined in Section II-B.

Regarding hyperparameters, we establish three configurations: *throughput-optimized*, *balanced*, and *accuracy-oriented*. The balanced configuration aligns with default settings from reference implementations whenever possible, while the other configurations adjust computational resources to balance throughput and quality. Table II summarizes these hyperparameters for each component.

Scope and limitations. This synthetic scenario aims to characterize system performance comprehensively. It captures size-driven impacts and heterogeneity across pipeline components, spanning representative computational conditions under controlled variations in protein and ligand sizes. However, it does not evaluate biochemical compatibility or the experimental viability of specific scaffold-ligand pairs.

TABLE II
COMPONENTS AND HYPERPARAMETERS. THR./BAL./ACC. DENOTE THROUGHPUT-OPTIMIZED, BALANCED, AND ACCURACY-ORIENTED.

Component	Hyperparameter	Thr.	Bal.	Acc.
RFdiffusion	`num_steps`	20	50	100
ProteinMPNN	–	–	–	–
ESM-2	`model_size`	150M	650M	3B
MMseqs2	`max_iterations`	–	3	5
Protenix	`num_steps`	100	200	300
Protenix	`num_cycles`	2	4	8
ESMFold	`num_cycles`	2	4	8
AlphaFold3	`num_steps`	100	200	300
AlphaFold3	`num_cycles`	5	10	15
DiffDock	`num_steps`	10	20	30
Vina-CPU	`exhaustiveness`	4	8	16
Vina-GPU	–	–	–	–

B. Implementation Details

Software setup. Each component is containerized with Docker to ensure reproducibility and portability. Pipeline workflows are orchestrated using Nextflow [15], a widely adopted bioinformatics workflow manager. Tasks are scheduled through Kubernetes, augmented by the Alibaba GPUShare plugin, which exposes GPU memory as a schedulable resource to prevent out-of-memory errors during concurrent operations. Empirically, we observe negligible performance impact from these orchestration layers given our workload characteristics. The software stack comprises Ubuntu 22.04, Nextflow 25.04, Docker 28.1, and Kubernetes 1.33.

Hardware setup. Experiments are conducted on a server equipped with one RTX 3090 GPU, two Intel Xeon Gold 6326 CPUs, 256 GB DRAM, and 4 TB PCIe Gen 4 NVMe SSD.

IV. PERFORMANCE CHARACTERIZATION

A. Component-Level Observations

Figure 2 profiles individual components of the protein-design pipeline across three operating points and nine scaffold–ligand pairs spanning different sequence lengths. We measure wall-clock time, temporal utilization, and peak device memory usage via the NVIDIA Management Library (NVML). To evaluate spatial utilization, we first aggregate kernels accounting for at least 90% of the execution time using Nsight Systems, then calculate the runtime-weighted average SM occupancy with Nsight Compute.

Inputs and hyperparameters. Input length significantly influences runtime across components. RFdiffusion exhibits a particularly sharp increase, with runtime scaling up to 5.1 $\times$ when scaffold length expands from 1SHG to 1TIM. Other components experience noticeable but smaller runtime increases. MMseqs2 shows variable runtime spikes driven by factors beyond sequence length alone, including database interactions. Hyperparameter adjustments consistently influence runtime, although input length generally has a more substantial effect. Accuracy-oriented configurations extend execution time compared to throughput-oriented settings.

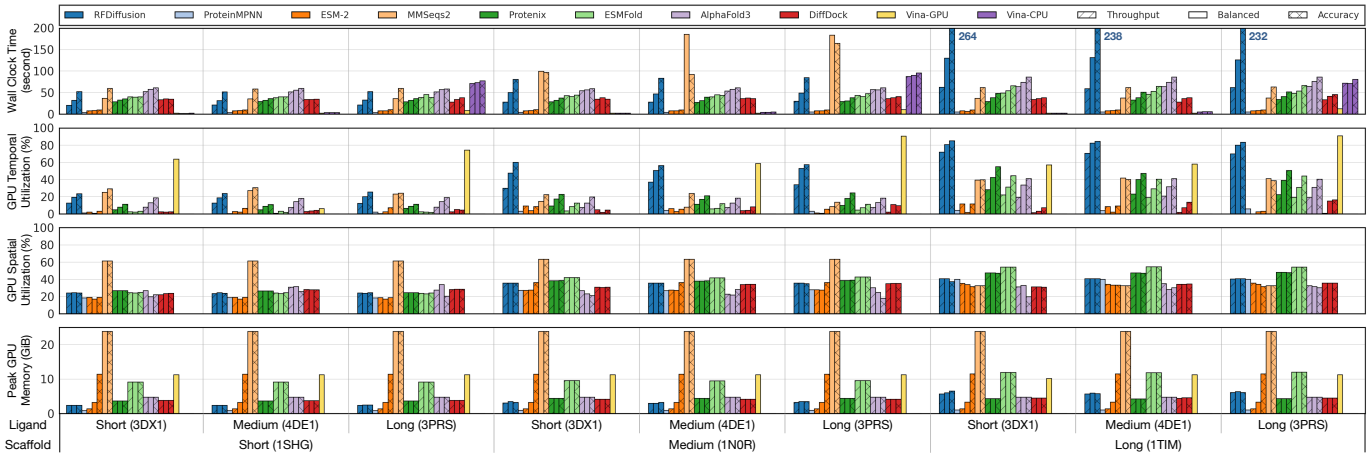

Fig. 2. Profiling of individual pipeline components across varying inputs and hyperparameters. Vina-CPU does not utilize GPU resources. Vina-GPU is implemented using OpenCL, which does not support spatial profiling; thus, only temporal utilization is reported.

GPU utilization. Temporal GPU utilization varies significantly across components. RFdiffusion and Vina-GPU achieve relatively high utilization. MMseqs2 shows moderate utilization around 26%. Vina-GPU utilization notably improves for larger ligands, reaching 70 to 91%. RFdiffusion similarly achieves higher utilization with longer sequences. ESMFold, Protenix, and AlphaFold 3 occasionally reach moderate utilization with larger inputs, yet low utilization dominates overall.

Spatial utilization is generally modest, predominantly clustering between 24% and 38%. Notable exceptions include ESMFold's large input scenario, which reaches approximately 54%, and MMseqs2, consistently higher at 61 to 63%. Experiments conducted on an RTX 3090 suggest the model sizes and execution parameters do not fully leverage the GPU's computational capacity, indicating room for optimization.

Peak GPU memory. Peak GPU memory usage remains consistently below device limits of 24 GB, except for MMseqs2, suggesting ample headroom for co-locating multiple processes.

B. Pipeline-Level Observations

We extend our analysis from isolated components to entire end-to-end pipelines, using the balanced configuration with medium-length scaffolds and ligands. For the structure prediction stage, we select Protenix, and for the validation stage, we use Vina-GPU.

Sampling for exploration. Several components offer sampling hyperparameters that expand the search space. RFdiffusion can produce multiple backbone structures per prompt, ProteinMPNN can generate multiple sequences per backbone, ESM models propose sequence variants, AlphaFold-style predictors return multiple structural candidates, and AutoDock evaluates several docking poses. When an upstream component outputs multiple candidates, these propagate downstream.

To illustrate the impact of sampling, we experiment with a uniform sampling configuration, setting sampling counts to 1, 2, or 3 for all components supporting sampling, producing 1, 16, and 81 samples, respectively. Figure 3 shows the normalized runtime breakdown for each case. With a single sample per component, the total runtime aligns with the sum of component-level runtimes shown in Figure 2. Reusing

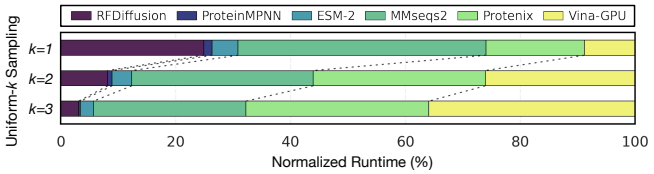

Fig. 3. Normalized runtime of pipeline components across varied sampling.

common outputs from early stages significantly improves exploration throughput, reducing per-sample latency from 4.32 minutes (uniform-1) to 1.18 minutes (uniform-2) and further down to 0.83 minutes (uniform-3). As sample counts increase, runtime shifts towards downstream components, reflecting multiplicative workload expansion.

GPU co-location. Figure 4(a) and (b) illustrate GPU temporal utilization with and without co-location on a single GPU, for uniform sampling counts of two and three, respectively. Co-location, in which multiple processes timeshare GPU resources, enhances temporal utilization and throughput, reducing overall execution time. This benefit grows with higher sampling counts, providing greater scheduling flexibility. Specifically, co-location decreases latency by 11.9% for uniform-2 sampling and by 27.7% for uniform-3 sampling. Later-stage components particularly benefit from co-location due to their lower peak memory requirements, allowing more concurrent executions.

Multi-GPU scaling. Figures 4(c) and (d) evaluate scaling to multiple GPUs using higher uniform sampling counts of four and five with co-location enabled. In early stages, limited sample availability causes low mean temporal utilization as many GPUs remain idle. Utilization gradually improves over time as more samples become available. At current uniform sampling levels, GPUs remain generally underutilized, resulting in weak scaling efficiency. Higher sampling counts lead to stronger scaling improvements. For example, scaling from one to two GPUs reduces latency by up to 42.8% with uniform-5 sampling but shows less improvement with uniform-4 sampling. Scaling efficiency noticeably declines at three and four GPUs with latency reductions reaching only 47.6% and 50.2%, respectively. Improving GPU utilization early in execution and achieving consistent scaling across

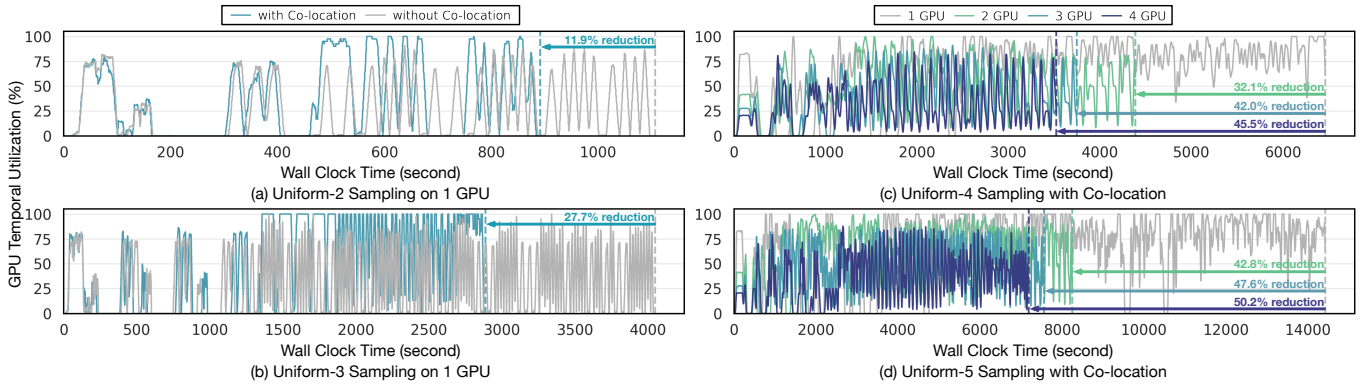

Fig. 4. GPU temporal utilization over execution time.

multiple GPUs are key considerations for future work.

V. Future Directions

Compute-cost analysis. Selecting among model families and hyperparameters shifts the Pareto frontier between accuracy and throughput. Choices like substituting larger ESM variants, using ESMFold or Protenix over alternative predictors, or altering docking seed policies impact both quality and runtime. Early exploration might prioritize rapid approximations, increasing fidelity later. Current measurements, however, are input-dependent, complicating comparisons across studies. We propose standardized metrics such as per-sample latency or energy per accepted candidate at defined quality thresholds.

Large-scale setups. Scaling pipelines introduces complexities as increased sampling or tasks like multimer predictions greatly expand computational demands. Deploying at scale with multiple GPUs raises challenges in resource placement and scheduling. Future work should examine packing strategies considering compute and memory jointly and enabling multi-GPU scheduling. Additional resources like CPU and storage may gain importance. Investigating interactions between partitioning methods (e.g., MIG, MPS) and length-bucketing strategies to reduce fragmentation is essential.

Agentic LLMs. Expanding candidate sets intensify the bottleneck of human oversight. Agentic language models (LLMs) can rank candidates, allocate computational resources adaptively, and justify decisions like escalation or early stopping. The Coscientist [16], demonstrating agent-driven scientific discovery, highlights similar orchestration roles in protein-design pipelines. Integrating LLMs as a new computational component necessitates analyzing their scheduling strategies, resource management policies, and computational behaviors within the existing pipeline infrastructure.

VI. Related Work

To our knowledge, system-level characterizations that profile the full protein design pipeline under realistic co-location are still limited. At the component level, most works are centered around structure prediction, including characterization [17], system-level optimization for inference [8] and training [18], and an ASIC design [19] to support longer sequences. Homology search has also seen substantial GPU acceleration [5]. Our study complements these efforts by emphasizing end-to-end behavior, resource sharing, and utilization patterns in complete pipelines.

VII. Conclusion

We present a reproducible GPU-centric characterization of protein-design pipelines, highlighting computational heterogeneity and GPU underutilization. Our findings emphasize GPU-aware resource management, cost-aware evaluation metrics, and scalability analyses in multi-GPU setups. By releasing open-source artifacts, we support future comparable studies and practical pipeline optimizations.

References

[1] J. Abramson *et al.*, "Accurate structure prediction of biomolecular interactions with alphafold 3," *Nature*, vol. 630, pp. 493–500, 2024.
[2] J. L. Watson *et al.*, "De novo design of protein structure and function with rfdiffusion," *Nature*, vol. 620, pp. 1089–1100, 2023.
[3] J. Dauparas *et al.*, "Robust deep learning–based protein sequence design using proteinmpnn," *Science*, vol. 378, pp. 49–56, 2022.
[4] Z. Lin *et al.*, "Evolutionary-scale prediction of atomic-level protein structure with a language model," *Science*, vol. 379, pp. 1123–1130, 2023.
[5] F. Kallenborn *et al.*, "Gpu-accelerated homology search with mmseqs2," *Nature Methods*, pp. 1–4, 2025.
[6] ByteDance AI4Science Team, "Protenix-advancing structure prediction through a comprehensive alphafold3 reproduction," *bioRxiv*, 2025.
[7] G. Corso *et al.*, "Diffdock: Diffusion steps, twists, and turns for molecular docking," in *ICLR*, 2023.
[8] M. Mirdita *et al.*, "Colabfold: making protein folding accessible to all," *Nature methods*, vol. 19, pp. 679–682, 2022.
[9] D. Listov *et al.*, "Opportunities and challenges in design and optimization of protein function," *Nature*, vol. 25, pp. 639–653, 2024.
[10] N. Raouraoua *et al.*, "Massivefold: unveiling alphafold's hidden potential with optimized and parallelized massive sampling," *Nature Computational Science*, vol. 4, no. 11, pp. 824–828, 2024.
[11] O. Trott and A. J. Olson, "Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading," *J. Comput. Chem.*, vol. 31, pp. 455–461, 2010.
[12] S. Tang *et al.*, "Vina-gpu 2.1: towards further optimizing docking speed and precision of autodock vina and its derivatives," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, 2024.
[13] K. I. Albanese *et al.*, "Computational protein design," *Nature Reviews Methods Primers*, vol. 5, p. 13, 2025.
[14] M. Su *et al.*, "Comparative assessment of scoring functions: the CASF-2016 update," *J. Chem. Inf. Model.*, vol. 59, pp. 895–913, 2018.
[15] P. Di Tommaso *et al.*, "Nextflow enables reproducible computational workflows," *Nature*, vol. 35, pp. 316–319, 2017.
[16] J. Gottweis *et al.*, "Towards an ai co-scientist," *arXiv*, 2025.
[17] J. Kim *et al.*, "Alphafold3 workload characterization: A comprehensive analysis of bottlenecks and performance scaling," in *IISWC*, 2025.
[18] H. La *et al.*, "Megafold: System-level optimizations for accelerating protein structure prediction models," *arXiv*, 2025.
[19] S. Han *et al.*, "Lightnobel: Improving sequence length limitation in protein structure prediction model via adaptive activation quantization," in *ISCA*, 2025, pp. 1940–1955.