AlphaFold3 Workload Characterization: A Comprehensive Analysis of Bottlenecks and Performance Scaling

Jinpyo Kim*† Mingi Kwon*
Jishen Zhao*
*University of California, San Diego, USA
†SK Hynix, South Korea
Email: {jik066, mik090, jzhao}@ucsd.edu

***Abstract*—AlphaFold3 is a state-of-the-art deep learning model for biomolecular structure prediction, introducing architectural changes and expanded input diversity beyond AlphaFold2. These enhancements, while enabling richer biological modeling, redefine the performance bottlenecks of the system. We profile AlphaFold3 on representative hardware platforms and analyze execution time contributions from Multiple Sequence Alignment (MSA) and inference phases. Our analysis identifies cache-related bottlenecks, limited thread scaling, and GPU initialization overheads as key performance constraints. These results underscore the need for execution strategies that are aware of hardware architecture and sensitive to input characteristics—especially for AlphaFold3. To support reproducible performance evaluation, we provide an open-source benchmark suite AFSysBench[1] with integrated profiling tools, enabling detailed performance analysis across diverse CPU and GPU configurations.**

***Index Terms*—AlphaFold3, deep learning inference, diffusion models, high-performance computing (HPC), multiple sequence alignment (MSA), performance characterization**

I. Introduction

AlphaFold [18] is a deep learning-based system that accurately predicts the three-dimensional structure of proteins from their amino acid sequences, widely used in structural biology, bioinformatics, and related fields. It represents a key development in computational biology, significantly advancing biological discovery.

While AlphaFold2 (AF2) [10] has traditionally been widely used, AlphaFold3 (AF3) [5] released in 2024 is a more recent and advanced version for biomolecular structure prediction and interaction analysis. Supporting a broader range of applications, AF3 introduces new capabilities beyond protein structure prediction to include DNA, RNA, ligands, and ions, and their interactions. The model architecture of AF3 also adopts new components, such as a Pairformer and a diffusion-based generative model, which enhance modeling capability and increase generality.

However, system-level performance impacts introduced by the development of the AF3 model architecture have not been thoroughly studied. First, previous performance characterizations of AF2 show that the pipeline is computationally intensive, particularly in the MSA stage and the GPU-based inference phase [10], [18]. Unlike AF2, the MSA generation stage in AF3 is more complex. For instance, supporting RNA as a new input requires searches against additional nucleotide databases using tools like nhmmer [20], despite a reduction in the primary protein database size. Second, in the inference stage, the AF3 architecture is substantially changed with the introduction of the Pairformer and the diffusion-based generative model. As a result, prior system optimizations targeting the Evoformer no longer directly apply. Finally, the iterative

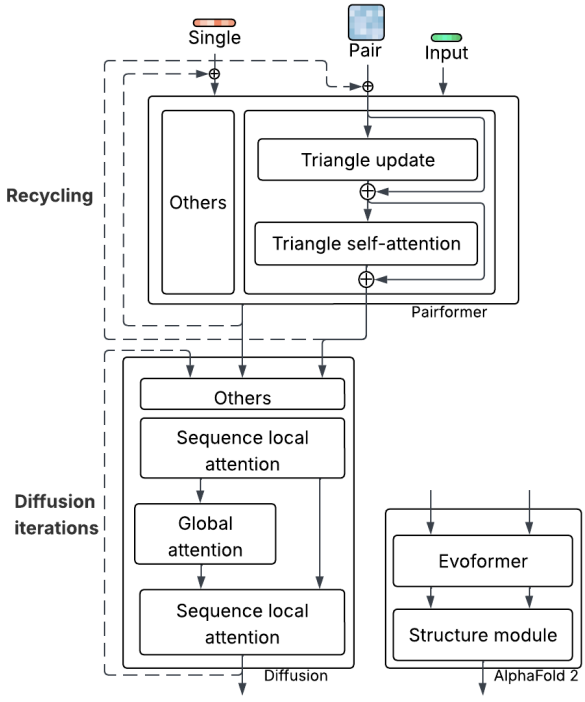


Fig. 1: Overview of AF2 and AF3 architectures. Our analysis focuses on the new Pairformer and Diffusion modules in AF3 (left), which replace AF2's Evoformer and Structure module (right). The diagrams are simplified to only show the main computational layers.

[1]Github repository: https://github.com/stable-lab/AFSysBench

nature of the diffusion model leads to recurrent memory access patterns not present in AF2, and further changes the characteristics of the computational workload. Therefore, a comprehensive analysis of performance characteristics and bottlenecks in the AF3 system is essential to enable efficient system resource utilization and software/hardware optimizations.

In this paper, we conduct a comprehensive system-level performance analysis of AF3 across both desktop and server system configurations with different CPUs and GPUs. We focus on two key stages: (i) the MSA phase, which prepares evolutionary features from homologous sequences, and (ii) the inference phase, which performs structure prediction using neural network modules—primarily the Pairformer and Diffusion module—to iteratively update pair representations that capture atomic interaction relationships. Our evaluation spans a diverse set of molecular inputs—from simple monomeric proteins to multi-chain nucleic acid complexes—capturing a representative range of use cases. By benchmarking end-to-end execution time across multiple samples and thread counts, we identify how performance is affected by factors such as molecular complexity and computer system architectures. Our study also allows us to correlate system-level inefficiencies with specific components in the AF3 pipeline, such as alignment search functions, data preparation steps, and neural network layers. In particular, we make the following key observations, with more detailed observations discussed in Sections IV and V.

- **The MSA phase is the primary performance bottleneck.** Counter-intuitively, despite a major redesign of the inference engine, the initial MSA generation stage dominates the end-to-end execution time. We observe that this phase accounts for 70-90% of the total runtime for complex inputs (Figure 3), with its scalability limited by CPU architectural configurations, such as L3 cache size (Table II) and memory-sensitive behavior in core alignment functions (Table IV).
- **Inference bottlenecks are highly system-dependent.** The composition of inference time varies dramatically across hardware platforms. On our server-class system (Intel Xeon with H100 GPU), initialization and Accelerated Linear Algebra (XLA) compilation consumed over 75% of the inference time for smaller inputs. In contrast, on our desktop-class system (AMD Ryzen with RTX 4080 GPU), the actual GPU computation was the dominant factor, highlighting a significant difference in the system balance between the two configurations (Figure 8).
- **Standard desktop hardware supports efficient AF3 execution.** We demonstrate that AF3 runs effectively on commodity hardware for many common use cases. Despite having a less powerful GPU, our desktop-class AMD Ryzen system successfully processed all test samples, including a 1K-residue protein complex. This suggests that a strong CPU is more critical for many workloads than a top-tier GPU, increasing AF3's accessibility.

Our study shows that AF3 introduces new performance characteristics absent in AF2 and that architecture-aware analysis is critical to understand and guide platform selection, resource allocation strategies, and computer system design for efficient real-world deployments. Beyond the characterization findings, we also provide an open-source benchmark suite, **AFSysBench**, with integrated profiling tools to support reproducible evaluation and practical optimizations.

II. Background and Motivation

A. Multiple Sequence Alignment (MSA)

Multiple Sequence Alignment (MSA) is a bioinformatics technique that aligns three or more biological sequences, such as proteins or nucleic acids, to find similar regions that suggest functional, structural, or evolutionary connections. The output is a representation of shape $(M \times N \times d)$, where M is the number of sequences, N is the aligned length, and d is the hidden feature dimension. These representations encode biological features such as sequence and positional information. Unlike AF2, AF3 significantly reduces the reliance on MSA representations. While MSA information is still used, its role is greatly diminished in favor of single and pair representations, which helps to simplify the model's architecture.

Pair Representation. Pair representation encodes features between every pair of residues or atoms, capturing spatial proximity, geometric constraints, and co-evolutionary signals. It is typically represented as a tensor of shape $(N \times N \times d)$, where each element contains relational information specific to the residue pair at the corresponding row and column indices—describing how two residues interact or relate within the protein structure.

Single Representation. The single representation refers to per-residue or per-atom embeddings that capture local biochemical properties. It is stored as a tensor of shape $(N \times d)$, where N is the number of residues and d is the feature dimension. AF3 updates this representation within the *Pairformer* module using self-attention and bias-attended mechanisms, which are abstracted as the *Others* block in Figure 1. These updates enable the integration of both local context and pairwise relational signals, which are subsequently fused during downstream refinement steps.

B. Pairformer

The **Pairformer** module in AF3 replaces the Evoformer stack in AF2 as the primary computational block. Unlike AF2, AF3 significantly reduces the usage of MSA representations, relying mostly on the single and pair representations. While the depth of the stack (48 blocks) remains similar to AF2, Pairformer processes only these two representations. The resulting pair and single embeddings, together with the input representation, are then passed to the newly introduced **Diffusion** module, which replaces the structure module of AF2.

Triangle (Multiplicative) Update. This layer refines each residue pair (i, j) by aggregating information from indirect triangle paths through all intermediates k. Two symmetric variants operate on outgoing edges ($i \rightarrow k, j \rightarrow k$) and

incoming edges ($k \to i, k \to j$). The update combines features (i, k) and (j, k) through element-wise multiplication followed by summation across all intermediates, e.g.,

$$z_{ij} = \sum_k z_{ik} \odot z_{jk},$$

providing a differentiable analog of the triangle inequality that enforces geometric consistency [10].

Triangle Self-Attention. This operation applies self-attention to pair representations in two modes: *starting node* (attending across outgoing edges from i) and *ending node* (across incoming edges to j). Attention logits are biased by the third edge in the triangle (e.g., $z_{j,k}$ or $z_{k,i}$), enabling the model to propagate global structural context beyond direct residue-residue interactions [10].

C. Diffusion Module

The diffusion module casts structure prediction as a generative process: starting from noisy atomic coordinates, it iteratively denoises using learned geometric priors. Each step updates both coordinates and features, requiring storage of intermediate states and incurring recurrent memory loads absent in AF2. This results in *higher per-sample memory usage* and reduced benefit from prior optimization techniques [5], [7].

Sequence-local Attention. Operates over local sequence windows to refine short-range interactions, essential for secondary structures and local folding patterns.

Global Attention. Captures long-range dependencies across chains and large motifs but scales quadratically with sequence length and suffers from poor memory locality, making it a key bottleneck in large complexes.

D. Motivation

AF3 [5] introduces a fundamentally different architecture from its predecessor **AF2** [10], including (i) replacing the Evoformer with the *Pairformer*, (ii) eliminating the use of MSA representation and processing solely on single and pair representations, and (iii) replacing the structure module with a *diffusion-based generative model* to predict 3D molecular conformations.

Prior studies have focused on examining the performance characteristics of AF2 inference workloads. For example, FastFold [6] identifies memory-bound attention layers in Evoformer as critical performance bottlenecks in AF2; the study accelerates them using dynamic axial parallelism (DAP) to reduce memory overhead and enhance parallelism by decoupling attention heads across axes.

However, these approaches are tightly coupled with the AF2 design with deep MSA stacks and joint attention across the MSA, single, and pairwise representations. The major design changes in AF3 invalidate the prior assumptions. In particular, the absence of MSA representation, the decoupling of structure generation into a diffusion module, and the exclusive use of single and pair representations significantly change the computational profile and memory access patterns. As a result, new bottlenecks emerge, while the existing AF2-based optimizations cannot be directly applied. To address this, this paper presents **the first** in-depth system-level characterization of AF3 inference workloads, offering insights for future software and hardware design efforts optimized for the new development in AF systems.

TABLE I: System Hardware Configurations.

Configuration	Server	Desktop
CPU	Intel Xeon Gold 5416S	AMD Ryzen 7900X
Core/Thread	16/32	12/24
Base Clock	2.0GHz	4.7GHz
Max Clock	4.0GHz	5.6GHz
L1/L2 Cache	80KB/2MB per core	64KB/1MB per core
Last Level Cache	30 MB shared	64 MB shared
Memory Type	DDR5-4400	DDR5-6000
Memory Size	512 GiB (32 GiB × 16)	64 GiB (32 GiB × 2)
Mem. Expander	CXL (256 GiB)*	–
GPU	H100 80 GB	RTX 4080 16GB
Storage	PCIe 4.0 SSD	PCIe 4.0 SSD

*Optional Memory Expansion Used Only in Sec III-C Experiments.

III. Methodology and Basic Analysis

This section presents our experimental setup, input characteristics, and our benchmark suite AFSysBench.

A. System Configurations

We evaluated AF3 on two platforms (Table I). The **Server** system, representative of typical HPC server configurations, features an Intel Xeon CPU, large memory capacity, and an NVIDIA H100 GPU. The **Desktop** system uses an AMD Ryzen CPU and an NVIDIA RTX 4080 GPU, offering higher CPU clock frequency but smaller memory and a consumer-grade GPU. We refer to these as Server and Desktop, respectively, to highlight differences between parallel HPC systems and single-core–optimized desktop architectures.

B. Input Format and Sample Characteristics

AF3 adopts input in a structured JSON format that defines the biomolecular sequences to be modeled, specifying chain composition and molecular types. A notable strength of AF3 is its capacity to process a wide range of biomolecular assemblies, from simple monomeric proteins to large, multi-component complexes. This flexible input schema allows AF3 to support a wide range of biological modeling tasks across different molecular modalities and complexities. In our evaluation, we select representative biomolecular systems – including monomeric proteins, protein-protein complexes, protein-RNA assemblies, and protein-DNA assemblies – to systematically assess AF3 inference behavior in varying structural compositions and input sizes. These inputs are encoded in the AF3 input format and evaluated on different hardware platforms. Table II summarizes the key properties of the selected inputs used in our benchmarks. Each MSA search accessed different protein and RNA sequence databases, depending on the molecule types involved.

To systematically analyze the performance of the AF3 pipeline, we curated a diverse suite of benchmarks in AFSysBenchsummarized in Table II. The selection was designed to enable direct comparisons between key workload characteristics. For instance, 2pv7.json (protein-only)

TABLE II: Summary of Input Samples Used in AF3 Experiments.

Sample	Structure	Complexity	Seq. Length	Primary Benchmark Target / Workload Characteristic
2PV7	Protein (2 chains)	Low	484	Symmetric multi-chain processing
7RCE	Protein (1) + DNA (2)	Low-Mid	306	Baseline for mixed-type input
1YY9	Protein (3 chains)	Mid	881	Asymmetric multi-chain complex
Promo	Protein (3) + DNA (2)	Mid-High	857	MSA pipeline stress with low-complexity sequence
6QNR	Protein (9) + RNA (1)	High	1,395	High chain-count assembly with mixed input types

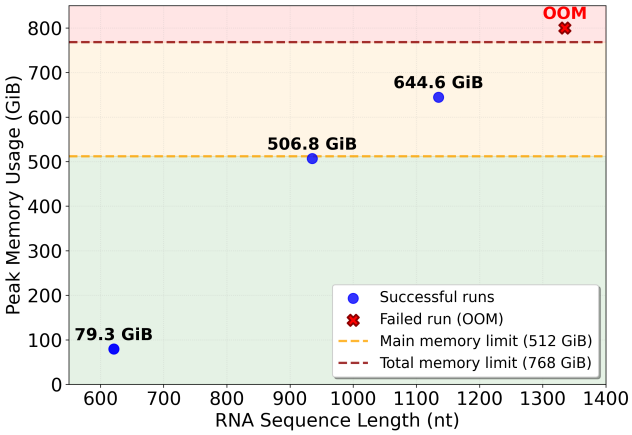


Fig. 2: Peak memory consumption as a function of RNA sequence length. The horizontal lines indicate the main memory and total memory with CXL expansion.

and rcsb_pdb_7rce.json (protein-DNA), both having short sequence lengths, were chosen to isolate the performance impact of including DNA chains alongside a protein. Similarly, the 1YY9.json input, a standard multi-chain protein system, serves as a baseline for comparison against promo.json. This promo.json sample was specifically selected because it is known to contain a low-complexity sequence, allowing us to stress-test the MSA search and filtering stages. Finally, 6QNR_subset.json represents a significant step up in complexity, incorporating a large number of protein chains plus an RNA chain. This benchmark suite allows us to observe how system scalability and performance characteristics change when handling a complex, high-chain-count, mixed-molecule assembly.

Notably, the 6QNR sample required a DRAM upgrade to 128 GiB DDR4-3600MHz on the Desktop system due to an out-of-memory (OOM) failure in the default configuration. In addition, the inference phase for 6QNR exceeded the memory capacity of the RTX 4080 GPU (16GB), necessitating the use of AF3's unified memory option to offload excess data to CPU memory. These inputs enable systematic characterization of AF3's performance across a broad spectrum of molecular complexities and input configurations.

C. Analysis of CPU Memory Requirements

The memory requirements of AF3, particularly during the MSA generation stage for RNA sequences, are sensitive to input characteristics. These include the molecular composition, long RNA chains, and the total residue count. Notably, peak memory consumption for long RNA inputs is largely independent of thread count. AF3 does not perform static memory validation to check whether the input will exceed system memory capacity. As a result, when memory requirements exceed available system resources, the process may terminate unexpectedly, either due to operating system intervention (e.g., OOM kill) or internal allocation failure.

We observed that memory consumption of `nhmmer` increased non-linearly with RNA input length derived from the 7K00 ribosomal complex. As shown in Figure 2, the 621-residue RNA was processed in 79.3 GiB of memory, whereas increasing the length to 935 residues resulted in memory usage of 506 GiB. The 1,135-residue input consumed 644 GiB and completed only on the Server environment with CXL memory expansion [1]. Attempts to process the 1,335-residue input failed, exceeding the memory capacity of 768 GiB.

For comparison, protein-only inputs showed significantly lower memory footprints. A 1,000-residue protein chain consumed approximately 0.9 GiB with 8 threads, while a 2,000-residue input used around 1.7 GiB. Notably, the same 1,000-residue input required only 0.23 GiB when executed with a single thread, highlighting that memory usage also scales with thread count. Importantly, we found that the number and length of accompanying protein chains had negligible impact on peak memory usage. These results indicate that the memory footprint of `nhmmer` for long RNA inputs dominates overall memory consumption.

D. AFSysBench *Benchmark Suite*

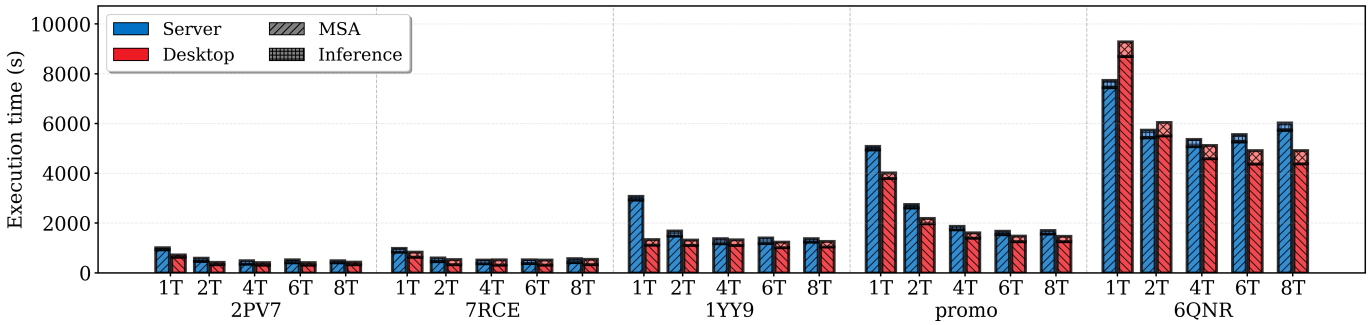
To support reproducible performance measurements, we developed a shell-script-based benchmark suite tailored for evaluating AF3. The suite builds on the official AF3 Docker-based installation and provides the following capabilities:

- Automated sequential execution of multiple input samples through MSA and inference stages.
- Thread scaling experiments (1, 2, 4, 6 and 8 threads).
- Support for diverse input configurations, including various sequence lengths and molecular types (protein, RNA, and DNA complexes).
- Integration of profiling tools: `perf`, AMD uProf, `iostat`, and NVIDIA Nsight Systems.

IV. System-Level Performance Overview

A. Server vs. Desktop Performance

Figure 3 presents the total execution time of AF3 on two hardware platforms — Server and Desktop — as a function of thread count. Five representative samples (2PV7, 7RCE, 1YY9, promo, and 6QNR) are evaluated. The stacked bars



*The maximum coefficient of variation (CV) was within 5% for MSA (Server $\leq$ 5%, Desktop $\leq$ 2%) and within 1% for inference on both platforms.

Fig. 3: Total AF3 execution time across four input samples and two hardware platforms. Stacked bars indicate the relative contributions of MSA and inference phases under varying thread counts.

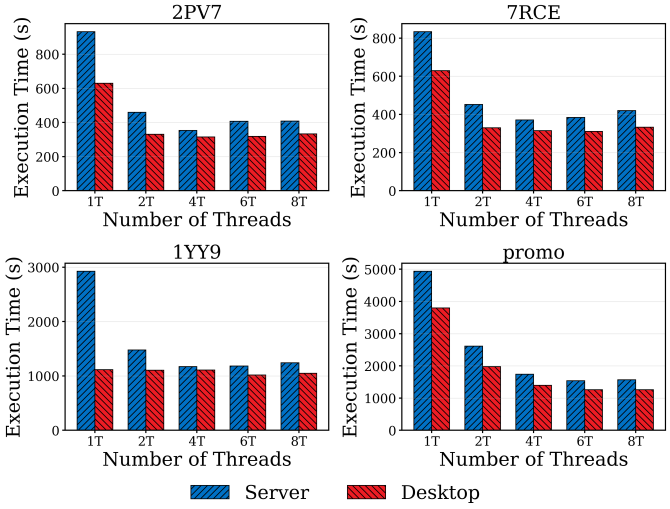


Fig. 4: MSA execution time on different samples and hardware platforms across 1–8 threads.

separately show the contribution of the MSA phase and the inference phase, highlighting how runtime varies with input sequence complexity and parallel thread configuration. The measurements are stable across five runs.

Overall, these results demonstrate that Desktop systems can deliver surprisingly strong performance for AF3 workloads. In particular, they consistently achieve shorter end-to-end execution times than Server, largely due to higher CPU clock speeds and superior memory efficiency in the CPU-bound MSA phase. Mid-scale inputs (up to roughly 1,000 residues) do not exhibit memory bottlenecks on the Desktop, making it a cost-effective and practical alternative to expensive Server. However, larger-scale or highly complex workloads may still require server-class resources to avoid performance degradation.

Observation 1. Consumer-grade systems can efficiently support AF3 inference on moderately sized inputs, offering a cost-effective alternative to server-class platforms.

B. Input-Specific Runtime Characteristics

The promo exhibits significantly longer MSA runtime than 1YY9, despite having similar residue lengths. This is primarily due to poly-glutamine (poly-Q) repeats in chain A of promo, which result in excessive partial matches during database searches. The repetitive nature of these poly-Q stretches produces many low-complexity regions, causing jackhmmer to report a large number of ambiguous or partial alignments that still must be scored and filtered, thereby increasing search time. In contrast, 1YY9 contains more diverse and complex domains, resulting in a smaller set of higher-confidence candidate hits that can be processed more efficiently. The additional DNA chains in promo are excluded from the MSA phase and therefore do not affect its execution time.

Observation 2. Sequence composition can significantly prolong MSA runtime despite similar residue lengths, highlighting the critical impact of input characteristics on performance.

C. Detailed Phase Breakdown

1) MSA Phase Benchmark - Thread Scaling and Efficiency: Figure 4 shows the MSA execution time as a function of thread count from 1 to 8. The observed scaling behavior is non-linear and depends on sequence complexity. Moving from one to two threads yields near-ideal speedup (approximately 2×) across all samples. However, speedup gains diminish beyond four threads.

For smaller samples (e.g., 2PV7, 7RCE), performance slightly degrades beyond 4 threads. In contrast, larger samples (e.g., 1YY9, promo) benefit from up to 6–8 threads, although speedup gains saturate. Optimal thread count depends on both the input size and the hardware configuration. Four threads are generally sufficient for small samples, while six threads better serve larger ones. These results indicate that static threading policies are suboptimal. We recommend adaptive thread allocation based on input complexity and hardware configuration.

Figure 5 provides a breakdown for 6QNR, the most compute-intensive sample in our dataset. The left panel shows a steep speedup from 1 to 2 threads, followed by dimin-

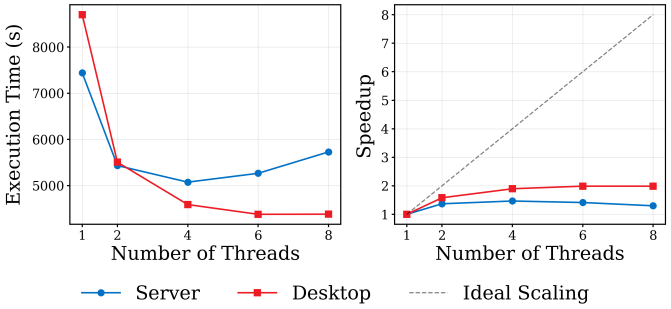


Fig. 5: Thread-level performance and speedup scaling of MSA on the 6QNR sample.

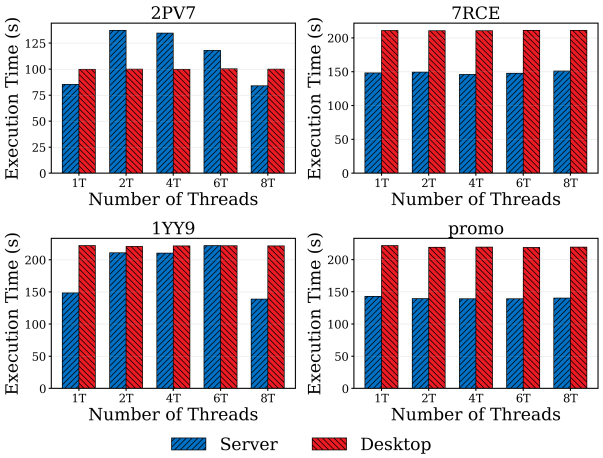


Fig. 6: Inference phase execution time comparison across different thread configurations for four protein samples on Server and Desktop systems.

ishing returns beyond 4 threads. The right panel illustrates a clear departure from linear scaling, highlighting saturation effects. Notably, we observe performance degradation beyond 4 threads, where execution time increases at 6 and 8 threads. Given that AF3 defaults to 8 threads, this fixed setting may be inefficient depending on input characteristics and system configuration.

2) Inference Phase Benchmark - Thread Scaling and GPU Utilization: We evaluate inference scalability across CPU–GPU systems by varying thread count from 1 to 6. As shown in Figure 6, inference performance shows minimal gains or even slowdowns with increasing threads. On Server system, small inputs (e.g., 2PV7, 1YY9) experience degraded performance under multi-threading, while larger inputs (e.g., promo) show only marginal improvements. Desktop system exhibits similarly flat scaling across all samples.

In summary, the inference phase exhibits limited scalability with multi-threading and substantial variation in GPU utilization across platforms. A deeper analysis of this behavior is provided in Section V-B3.

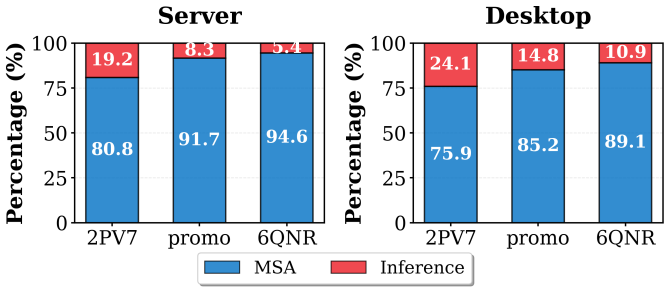


Fig. 7: Relative time distribution between both stages across different input and hardware configurations.

Observation 3. Input complexity and system-level bottlenecks such as cache and memory contention limit thread scalability, suggesting the need for adaptive, workload-aware allocation.

V. System-Level Bottleneck Characterization

While Section IV analyzed execution time trends across hardware and thread configurations, this section investigates the underlying bottlenecks driving those trends. We focus on phase-specific behavior and architecture-dependent inefficiencies not captured by aggregate runtime metrics.

A. Profiling Setup and Tools

We employed a multi-layered profiling approach to capture detailed system behavior.

- **CPU Performance**: `perf` for instruction-level metrics, cache hierarchy behavior, and branch prediction accuracy. `uProf` for AMD CPU L3 cache behavior.
- **Storage IO**: `iostat` to monitor disk-level load.
- **GPU Profiling**: NVIDIA Nsight Systems (nsys) and JAX Profiler to capture GPU kernel scheduling behavior.

This setup allows us to identify phase-specific characteristics and system bottlenecks that affect end-to-end performance.

B. Profiling-Based Bottleneck Analysis

1) MSA-to-Inference Ratio: Pipeline Composition: As observed in Figure 3, we examine the proportional contribution of the MSA and inference phases across different samples and hardware systems. Figure 7 shows the phase breakdown under optimal thread settings for each system. The results clearly indicate that the MSA phase overwhelmingly dominates the total execution time. Its contribution ranges from approximately 75–80% for simpler inputs to over 94% on the Server system for the most complex sample. While inference shares are slightly higher on Desktop systems, MSA remains the dominant phase across all configurations. These findings motivate a deeper investigation of CPU behavior and inefficiencies in the MSA phase, which we present in Section V-B2.

2) MSA Bottlenecks Across Architectures:

a) Cross-Architecture and Thread Scaling Behavior: We analyze the execution characteristics of the `2PV7` workload across two CPU architectures: Intel Xeon and AMD Ryzen.

Despite AMD's substantial frequency advantage, its wall-clock time at 4 threads is only modestly shorter than Intel's,

TABLE III: CPU Performance Metrics Comparison Across Samples and Thread Counts.

Input	Metric	Intel Xeon 1T	Intel Xeon 4T	Intel Xeon 6T	AMD Ryzen 1T	AMD Ryzen 4T	AMD Ryzen 6T
2PV7	IPC	3.68	3.56	3.49	3.08	2.91	2.85
	Cache Miss	17.4	30.9	41.0	15.1	13.1	12.4
	L1 Miss (%)	0.14	0.16	0.15	0.68	0.87	0.86
	LLC Miss (%)	56.2	55.6	56.4	1.1	6.3	41.4
	dTLB Miss (%)	0.01	0.01	0.01	20.1	35.7	37.0
	Branch Miss (%)	0.22	0.22	0.22	0.89	0.96	0.96
promo	IPC	3.34	3.39	3.40	2.99	2.77	2.48
	Cache Miss	33.3	31.9	35.6	5.31	4.85	4.14
	L1 Miss (%)	0.47	0.47	0.47	1.75	1.94	2.45
	LLC Miss (%)	59.6	55.5	38.6	26.3	26.3	19.0
	dTLB Miss (%)	0.00	0.00	0.01	6.55	11.9	10.4
	Branch Miss (%)	0.30	0.30	0.30	0.88	0.89	0.91

reflecting Intel's high per-cycle efficiency. Intel sustains a higher IPC and benefits from very low branch misprediction and negligible dTLB misses, attributes that reduce stalls and allow better utilization of its lower clock and smaller cache. In contrast, AMD benefits from a larger and more effective cache hierarchy, which initially yields low miss rates under single-threaded execution. However, Intel's smaller LLC is quickly overwhelmed, leading to frequent off-chip access. As the thread count increases, Intel's LLC miss rate remains flat, while AMD's grows markedly, indicating capacity saturation.

In summary, AMD follows a memory-centric strategy that leverages high frequency and large LLC to reduce effective memory latency. Intel, on the other hand, adopts a compute-centric design, relying on highly optimized pipelines and address translation mechanisms to maximize instruction throughput. The limited thread scalability in both systems arises from different bottlenecks: cache contention in AMD and persistent off-chip memory pressure in Intel. These findings emphasize the importance of memory hierarchy balance for memory-intensive workloads like MSA.

However, this thread-scaling behavior is not universal. For example, the `promo` sample—characterized by repetitive "Poly-Q" sequences—exhibits improved cache locality under multi-threading on the Intel platform. Unlike `2PV7`, where threading increases cache contention, `promo` on Intel benefits from parallelism: its LLC miss rate drops from 59.6% (1T) to 38.6% (6T), while IPC remains stable. We attribute this to regular access patterns that align well with hardware prefetchers, increasing memory-level parallelism as threads scale.

This contrast demonstrates that thread scalability depends not only on hardware features but also on the structure of the input data. While some inputs suffer from cache interference and memory pressure, others leverage predictable access patterns to make better use of available resources. A thorough performance evaluation must therefore account for both the architecture and input characteristics.

b) Function-Level Profiling: To identify the internal structure of the MSA workload, we conducted detailed function-level profiling using Linux perf record. Table IV presents the top CPU-consuming functions and their behavior

TABLE IV: Function-level Performance on Server.

Metric	Function	2PV7 1T	2PV7 4T	promo 1T	promo 4T
CPU Cycles (%)	calc_band_9	28.7	27.05	32.1	29.8
	calc_band_10	26.29	25.98	24.5	26.2
	addbuf	16.34	17.40	18.2	19.1
	seebuf	6.09	6.07	7.3	6.9
Cache Misses (%)	copy_to_iter	46.47	24.51	42.1	22.8
	calc_band_9	14.24	27.02	16.8	29.3
	addbuf	10.02	17.28	12.4	18.9

changes between single-thread and quad-thread execution for the 2PV7 sample.

The results categorize MSA workload into three key function types. **Computational functions** such as `calc_band_9` and `calc_band_10` dominate CPU cycles—collectively consuming 55%—and implement JackHMMER's core sequence alignment functionality. **I/O and parsing operations**, including `copy_to_iter`, exhibit high sensitivity to thread count. This kernel function—responsible for transferring data from disk to user program memory space—exhibits a reduction in cache miss contribution from 46.47% to 24.51% when scaling from 1 to 4 threads. This suggests improved I/O efficiency through parallelization. **Data buffering functions** like `addbuf` and `seebuf` together consume 23% of CPU cycles and handle input preparation and buffer lookahead logic. Their execution time remains stable under multi-threading.

Conversely, `calc_band_9`, which dominates the compute cycles during MSA, shows a significant increase in cache miss ratio (14.24% $\rightarrow$ 27.02%) under multi-threading. This suggests that although the function is computationally intensive, its performance becomes increasingly constrained by memory subsystem pressure as the number of threads grows. In particular, cache contention and reduced effective memory bandwidth lead to higher latency for data retrieval, stalling execution.

These findings highlight a key insight: even workloads that are compute-dominant—such as HMMER-based MSA alignment—are nonetheless subject to memory hierarchy bottlenecks. The high proportion of LLC accesses (e.g., 39% from `copy_to_iter`) and the observed cache miss trends confirm that efficient memory usage and cache locality are critical to sustaining performance under multi-core execution.

Observation 4. MSA execution is dominated by compute-intensive alignment functions, while thread scalability is limited by increasing cache contention. Higher cache and LLC miss rates point to memory hierarchy stress.

However, this behavior does not generalize uniformly across all inputs and platforms. Our function-level statistics reveal input- and system-specific shifts in bottlenecks:

2PV7. When scaling from 1 to 4 threads, `copy_to_iter` benefits from parallelism as cache misses drop nearly by half. However, `calc_band_9` emerges as the new bottleneck, with cache miss rates roughly doubling—marking a transition from compute-bound to memory-bound execution. On AMD,

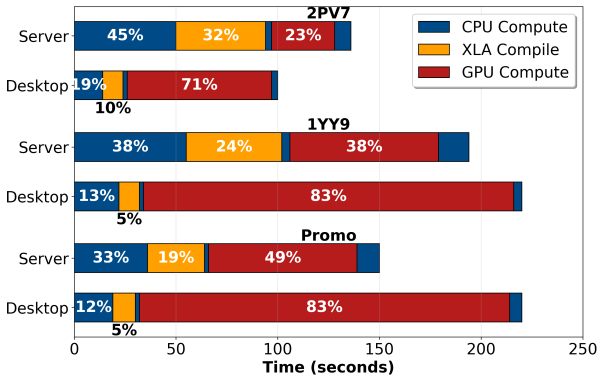


Fig. 8: Breakdown GPU Inference Time with Nsight Profiling on Different System.

this is observed as considerable L3 cache contention (rising from 1% to over 40%), while Intel shows cache inefficiency across the hierarchy.

promo. In contrast, repetitive sequence patterns improve cache locality. On Intel, 6-thread execution significantly reduces LLC misses while sustaining high IPC, enabling effective scaling. On AMD, cache misses improve only slightly, but IPC drops due to an increase in dTLB misses within `calc_band`. (observed in our AMD function-level profiling, not shown in Table IV). Compared to 2PV7, dTLB pressure is better managed because repetitive access patterns alleviate translation overhead.

c) Storage I/O Behavior and System Bottlenecks: We analyzed system-level disk activity using `iostat` during the `promo` and `6QNR` workloads to evaluate how storage access behavior impacts overall performance under varying memory constraints.

Server System: CPU-Bound Environment. Across both workloads, the Server system showed minimal disk activity, largely attributable to its 512GiB memory capacity, which allowed the large-scale databases to remain resident in DRAM via the page cache. Even during the RNA MSA phase of the `6QNR` workload—requiring access to an 89GiB RNA database not previously cached—storage activity remained low. NVMe SSD utilization rarely exceeded 20%. These observations indicate that storage was not a limiting factor. Instead, the longer execution time reflects a compute-bound scenario where performance is primarily limited by CPU throughput.

Desktop System: High-Throughput I/O-Bound. In contrast, the Desktop system—with 64 GiB of DRAM—was unable to retain the full dataset in memory, leading to frequent disk access. During peak execution phases, the primary NVMe SSD reached 100% utilization, indicating a clear I/O bottleneck. We observe that average read latency (`r_await`) remained consistently low (0.1–0.2 ms), indicating that the storage subsystem maintained high throughput under continuous load. This behavior reflects both the parallelism of the NVMe device and the CPU's ability to continuously process incoming data. Despite frequent disk access due to limited memory capacity, the system achieves efficient execution without observable degradation.

TABLE V: Inference Performance Bottlenecks on the Server.

Event Type	Function/Symbol	Sample	Overhead
Page Faults	*std::vector::_M_fill_insert*	2PV7	12.99%
		promo	16.83%
dTLB Load Misses	*xla::ShapeUtil::ByteSizeOf*	2PV7	5.99%
		promo	3.89%
LLC Load Misses	*copy_to_iter*	2PV7	6.90%
		6QNR	5.80%

3) Inference Phase Bottlenecks:

a) GPU Initialization Overhead Analysis: Figure 8 presents a time breakdown of the inference phase, profiled with NVIDIA Nsight Systems, revealing different performance characteristics between Server and Desktop platforms.

On the Server system, shorter input sequences show that CPU-side initialization and compilation dominate inference time. For 2PV7, the majority of execution is spent on GPU initialization and XLA compilation rather than computation. In 1YY9 and promo over half of the runtime is consumed by initialization, compilation, and finalization, leaving less than half for actual GPU computation. The Desktop system shows minimal overhead across all inputs. For 2PV7, GPU compute takes 71 seconds. XLA compilation adds 10 seconds, and CPU-side initialization and finalization take another 19 seconds. This trend continues with longer sequences, with GPU computation accounting for up to 83% of total runtime in both 1YY9 and promo. Inference bottlenecks vary with CPU–GPU coordination, leading to distinct performance behaviors across systems. On the Server, CPU-side overheads such as GPU initialization and XLA compilation dominate for short inputs. In contrast, the Desktop shows higher GPU computation time due to the relatively lower throughput of the RTX 4080 compared to the H100. Nsight profiling further revealed that CPU-to-GPU kernel dispatch is handled by a single host thread, explaining why multi-threading provided no performance benefit. We further profiled GPU initialization and XLA compilation phase on the Server using perf to identify the root causes.

b) Memory Allocation Bottleneck Analysis: Table V presents the major performance bottlenecks observed during the GPU initialization phase. The primary source of overhead is XLA's preparation phase, which repeatedly invokes `_M_fill_insert` to allocate and initialize large tensors. These allocations lead to frequent page faults (12–17%), increasing OS overhead and exposing memory subsystem inefficiencies. First, **dTLB-load-misses** are observed in `ByteSizeOf`, which calculates the required memory size based on tensor shape metadata. The resulting non-contiguous memory accesses account for 4-6% overhead. Second, **LLC-load-misses** appear in `copy_to_iter`, which handles data transfer from disk to user space, contributing 6-7% overhead.

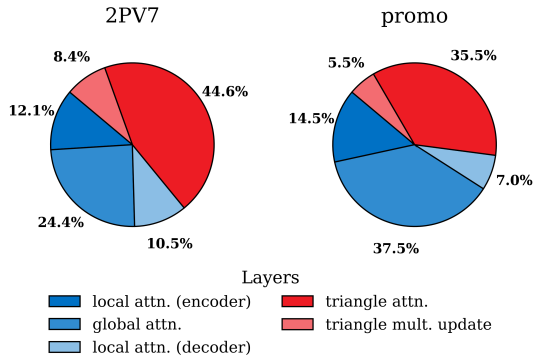


Fig. 9: Execution time breakdown of Pairformer (red slices: triangle attention and multiplicative update) and Diffusion (blue slices: local/global/decoder attention) layers.

C. Pairformer and Diffusion Module Analysis

AF3 introduces two novel architectural modules **Pairformer** and the **Diffusion module** to replace the Evoformer stack used in AF2. These new modules are responsible for updating both pairwise and single representations in a more flexible and generalizable manner, allowing AF3 to support heterogeneous biomolecular assemblies. Despite the architectural complexity of these modules, our performance profiling reveals that only a small subset of their individual layers contribute meaningfully to the overall execution time.

Specifically, among the many layers present in both Pairformer and Diffusion, we found that the **triangle multiplicative update** and **triangle attention** layers are the only ones with runtime impact within the Pairformer. In the case of the Diffusion module, significant compute time is dominated by the **global attention**, **local attention (encoder)**, and **local attention (decoder)** layers. These findings form the foundation for the deeper analysis in the following subsections: Section V-C1 discusses computational bottlenecks and complexity in the Pairformer, while Section V-C2 focuses on the iterative structure and performance cost of the Diffusion module.

1) Pairformer Performance Characteristics:

a) Triangle Attention and Multiplicative Update Complexity: The **triangle multiplicative update** layer updates pair representations via matrix multiplications or similar tensor operations, which become increasingly costly as sequence length grows, while the **triangle attention** layer aggregates triplet relationships by computing attention scores for each pair (i, j) over all possible intermediates (k), introducing a cubic $O(N^3)$ scaling bottleneck. For each of the N^2 residue pairs, the model must iterate over N possible intermediates, yielding a total computational cost proportional to N^3, which can be expressed as

$$Z_{ij} = \sum_{k=1}^{N} \text{Attention}(Q_{ij}, K_{ik}, V_{kj}) \,,$$

where

- Q_{ij} is the *query* vector from the residue pair (i, j),
- K_{ik} is the *key* vector from the residue pair (i, k),
- V_{kj} is the *value* vector from the residue pair (k, j).

For each pair of residues (i, j), the attention operation aggregates information over all possible intermediate residues k, resulting in N^2 pairs each considering N intermediates, and thus a total computational complexity of $O(N^3)$.

This analytical model explains why triangle attention becomes the dominant bottleneck as N increases, a trend confirmed by empirical results: as shown in Table VI, the runtime for the promo sample ($N = 857$) is more than three times that of the 2PV7 sample ($N = 484$), even though promo's sequence is only about 1.8× longer (Table II), demonstrating the superlinear growth consistent with cubic complexity.

b) Execution Time Distribution of Pairformer Module: As summarized in Figure 9, triangle layers account for a substantial portion of Pairformer runtime. For the `2PV7` sample, the triangle multiplicative update takes **8.4%** and triangle attention contributes **44.6%**, totaling **53.0%**. For the `promo` sample, these layers consume **5.5%** and **35.5%**, respectively, totaling **41.0%**. These results confirm triangle attention as dominant performance hotspots within the Pairformer.

2) Diffusion Performance Characteristics:

a) Structure Refinement via Iterative Denoising: The **Diffusion module** in AF3 refines 3D coordinates by applying learned denoising steps iteratively. Unlike traditional one-pass prediction, this repeated application of attention-based updates across 8 to 16 steps leads to a cumulative compute cost that grows linearly with the number of iterations, making it a performance-critical component.

b) Execution Time Distribution of Diffusion Module: Figure 9 highlights that within the Diffusion module, global attention is the dominant bottleneck. For the `2PV7` sample, global attention contributes **24.4%** of execution time, making it the single largest Diffusion component. For the `promo` sample, its share rises sharply to **37.5%**, outweighing all other Diffusion layers combined. These results confirm that the scalability and efficiency of global attention largely determine the overall performance characteristics of the Diffusion module.

3) Sequence Length and Composition Impact:

a) Observation on Sequence Length: Table VI shows that execution time grows substantially as the sequence length increases from 484 residues in 2PV7 to 857 residues in promo as summarized in Table II. Although promo is only about 1.8× longer, the Pairformer runtime expands by more than 3×, mainly driven by the triangle attention layer. Specifically, triangle attention grows from 21.23% of Pairformer time in 2PV7 to 32.58% in promo, reflecting its cubic $O(N^3)$ scaling. This clearly indicates that triangle attention becomes an increasingly dominant bottleneck as sequence length grows.

b) Observation on Heterogeneous Molecules: In addition to sequence length, promo's heterogeneous molecular composition—including three protein chains and two DNA chains—further contributes to increased triangle attention costs. Modeling pairwise relationships across diverse biomolecular types, such as protein–DNA interactions, increases relational complexity beyond that of a purely protein-based system like 2PV7. Consequently, the combination of

TABLE VI: Layer-Wise Execution Time Breakdown from JAX Profiler, Separated by Pairformer and Diffusion Modules.

Input	2PV7 (ms)	promo (ms)
Pairformer	15.87	53.19
triangle mult. update	4.03	12.03
triangle attention	8.14	31.09
Diffusion	80.37	147.53
local attn. (encoder)	12.49	20.15
local attn. (decoder)	10.00	15.88
global attention	53.08	102.64

longer sequences and greater molecular diversity results in both higher absolute and relative execution costs for triangle attention in promo.

VI. Discussions

Our characterization of AF3 exposes a variety of system-level inefficiencies across both the MSA and inference phases. Below, we outline several potential optimization directions to mitigate these issues and improve runtime performance.

Memory Estimation Based on Input Features. To improve execution reliability and avoid wasted computation, integrating a static memory estimator that analyzes input characteristics—particularly RNA length—prior to execution would be beneficial. This pre-check would help AF3 avoid unsafe configurations by issuing early warnings when inputs are projected to exceed memory capacity. Our analysis shows that long RNA chains can trigger non-linear spikes in memory usage, often exceeding system capacity.

Reducing GPU Initialization Overhead. As our analysis confirms, inference runtime is often dominated not by kernel execution but by JAX/XLA compilation and memory allocation. Under AlphaFold3's Docker-based runtime environment, each inference request incurs repeated model initialization, contributing to significant startup overhead. Avoiding this redundancy—such as by maintaining persistent model state—can substantially improve throughput and responsiveness.

Storage-Aware Optimization Strategies. We identify optimization in storage-aware design and propose two strategies to mitigate storage bottlenecks in MSA workloads.

(1) I/O Path Separation for Database Access. Since MSA workloads involve sequential scanning of large reference databases, we recommend isolating read-intensive operations—such as loading RNA or protein databases—onto high-throughput NVMe SSDs, ideally PCIe Gen5. optimized for sequential access. To minimize interference with alignment-critical database reads, auxiliary disk operations—such as logging and container metadata access—should be isolated to dedicated storage devices. This separation helps maintain consistent sequential read throughput for MSA workloads.

(2) Preloading Databases When memory capacity allows, preloading essential databases into DRAM before MSA execution can eliminate cold accesses during runtime. Rather than relying on demand-driven loading, a preprocessing stage could explicitly fetch and cache database files into memory using sequential read patterns, ensuring predictable performance. This approach is particularly effective on server-grade systems with a minimum of 512 GiB DRAM, where full database residency is feasible.

VII. Related Work

End-to-End AF2 Performance. While industry guides provide qualitative hardware recommendations for AF2 [2], [3], they do not offer a systematic analysis of execution bottlenecks. Recent research has targeted pipeline-level acceleration, such as ParaFold [21], which parallelizes MSA generation and structure inference across CPU and GPU to boost throughput. ScaleFold [22] focuses on AF2 training by applying kernel fusion, communication-aware scheduling, and end-to-end pipeline restructuring to optimize both computation and data movement across distributed systems. APACE [14] improves latency via a communication-efficient scheduler, while MassiveFold [17] scales inference across large GPU clusters. However, these works focus on orchestration rather than low-level architectural bottlenecks or CPU–GPU interactions.

MSA Bottlenecks. Multiple sequence alignment (MSA) remains a dominant bottleneck even with AF3's simplified MSA module. Tools like JackHMMER (for proteins) and nhmmer (for RNA) are retained for their sensitivity in detecting distant homologs [4], [12], [16], despite their computational cost. Optimizations include task parallelism [21], GPU-accelerated alternatives [11], [13], [15], RAM caching of sequence databases, and improved I/O pipelines [8]. Nonetheless, most studies treat MSA tools as black boxes, lacking microarchitectural insights into their thread scaling, cache, or memory behavior.

GPU Cold Start. ParaFold [21] and similar work examine redundant GPU JIT compilation overheads, proposing kernel reuse to accelerate repeated inference. Yet, first-request latency—critical for interactive workloads—remains largely unexplored. Related studies on serverless LLMs [9], [19] investigate GPU initialization, but do not analyze JAX/XLA-based pipelines where TLB misses, memory allocation, and data copies also impact latency.

VIII. Conclusion

This work presents a detailed performance study of AF3, analyzing both the MSA and inference phases. We find that the MSA phase dominates runtime, with scalability affected by cache behavior, architectural differences, and input-dependent threading sensitivity. The inference phase suffers from non-negligible GPU warm-up overheads. Our findings suggest that runtime efficiency in AF3 depends not only on architecture and cache hierarchy, but also on dynamic workload properties such as sequence length and alignment complexity.

Acknowledgment

This paper is supported by SRC JUMP 2.0 Center for Processing with Intelligent Storage and Memory (PRISM).

REFERENCES

[1] "CXL consortium releases compute express link 3.0 specification," Web, 2022, CXL Consortium; Accessed: June 30, 2025. [Online]. Available: https://www.computeexpresslink.org/pressroom

[2] "AlphaFold2 GPU benchmarking and hardware recommendations," Exxact Corporation, 2023, accessed: June 30, 2025. [Online]. Available: https://www.exxactcorp.com/blog/benchmarks/alphafold2-gpu-benchmarks-and-hardware-recommendations

[3] "AlphaFold2 NIM system requirements and performance," Web, 2023, NVIDIA; Accessed: June 30, 2025. [Online]. Available: https://docs.nvidia.com/nim/bionemo/alphafold2/latest/performance.html

[4] M. Abakarova, C. Marquet, M. Rera, B. Rost, and E. Laine, "Alignment-based protein mutational landscape prediction: doing more with less," *Genome Biology and Evolution*, vol. 15, no. 11, p. evad201, 11 2023.

[5] J. Abramson, J. Adler, J. Dunger, R. Evans, T. Green, A. Pritzel, O. Ronneberger, L. Willmore, A. J. Ballard, J. Bambrick, S. Bodenstein, D. A. Evans, C.-C. Hung, M. O'Neill, D. Reiman, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Accurate structure prediction of biomolecular interactions with AlphaFold 3," *Nature*, vol. 630, no. 8016, pp. 493–500, 2024.

[6] S. Cheng, X. Zhao, G. Lu, J. Fang, Z. Yu, T. Zheng, R. Wu, X. Zhang, J. Peng, and Y. You, "FastFold: Reducing AlphaFold training time from 11 days to 67 hours," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 28 256–28 268, arXiv:2203.00854v3 [cs.LG].

[7] G. Corso, H. Stärk, B. Jing, R. Barzilay, and T. Jaakkola, "DiffDock: diffusion steps, twists, and turns for molecular docking," in *International Conference on Learning Representations (ICLR)*, 2023, arXiv:2210.01776. [Online]. Available: https://openreview.net/forum?id=kKF8_K-mBbS

[8] H. Fujita, A. Nomura, T. Endo, and M. Sekijima, "Enhancing the performance of AlphaFold through modified storage method and optimization of HHblits on TSUBAME3.0 supercomputer," in *Proceedings of the 2023 Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE)*, 2023, pp. 2140–2146.

[9] H. Ghosh, "Enabling efficient serverless inference serving for LLM (large language model) in the cloud," *arXiv preprint arXiv:2411.15664*, 2024, accessed: June 30, 2025.

[10] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[11] F. Kallenborn, A. Chacon, C. Hundt, and M. Steinegger, "GPU-accelerated homology search with MMseqs2," *bioRxiv*, 2024, preprint; peer review pending.

[12] M. Mirdita, K. Schütze, Y. Moriwaki, L. Heo, S. Ovchinnikov, and M. Steinegger, "ColabFold: making protein folding accessible to all," *Nature Methods*, vol. 19, pp. 679–682, 2022.

[13] A. Müller, B. Schmidt, R. Membarth, R. Leißa, and S. Hack, "AnySeq/GPU: A novel approach for faster sequence alignment on GPUs," in *Proceedings of the 36th ACM International Conference on Supercomputing (ICS)*, 2022, pp. 20:1–20:11.

[14] J. Park, H. Jeong, J. W. Lee, and G. Kim, "APACE: Accelerating AlphaFold2 with parallel architecture and communication-efficient scheduling," in *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2024.

[15] S. Park, H. Kim, and T. Ahmad, "SaLoBa: Maximizing data locality and workload balance for fast sequence alignment on GPUs," *arXiv*, 2023.

[16] Z. Peng, W. Wang, H. Wei, X. Li, and J. Yang, "Improved protein structure prediction with trRosettaX2, AlphaFold2, and optimized MSAs in CASP15," *Proteins: Structure, Function, and Bioinformatics*, vol. 91, no. 11, pp. 1704–1711, 2023.

[17] N. Raouraoua, C. Mirabello, T. Véry, C. Blanchet, B. Wallner, M. F. Lensink, and G. Brysbaert, "MassiveFold: unveiling AlphaFold's hidden potential with optimized and parallelized massive sampling," *Nature Computational Science*, vol. 4, pp. 824–828, November 2024, received: 16 September 2024; Accepted: 03 October 2024; Published: 11 November 2024.

[18] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. R. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. T. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.

[19] Y. Shen, H. Ye, L. Tang, L. Zhang, J. M. Hellerstein, and I. Stoica, "Reducing the cost of GPU cold starts in serverless deep learning inference serving," in *Proceedings of the 2023 USENIX Annual Technical Conference (USENIX ATC '23)*. USENIX Association, 2023, pp. 115–129.

[20] T. J. Wheeler and S. R. Eddy, "nhmmer: DNA homology search with profile HMMs," *Bioinformatics*, vol. 29, no. 19, pp. 2487–2489, 2013.

[21] W. Zhang, X. Li, and F. Zhao, "ParaFold: A parallel and pipelined workflow for accelerating AlphaFold2 inference," in *Proceedings of the 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2021.

[22] F. Zhu, A. Nowaczynski, R. Li, J. Xin, Y. Song, M. Marcinkiewicz, S. B. Eryilmaz, J. Yang, and M. Andersch, "ScaleFold: reducing AlphaFold initial training time to 10 hours," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. Association for Computing Machinery, 2024, article 265, pp. 1–6.