

COSMIC: Enabling Full-Stack Co-Design and Optimization of Distributed Machine Learning Systems

Aditi Raju*

Harvard University
Cambridge, MA, USA
araju@college.harvard.edu

Jared Ni*

Harvard University
Cambridge, MA, USA
jaredni@college.harvard.edu

William Won*

Georgia Institute of Technology
Atlanta, GA, USA
william.won@gatech.edu

Changhai Man*

Georgia Institute of Technology
Atlanta, GA, USA
cman8@gatech.edu

Srivatsan Krishnan*

Harvard University
Cambridge, MA, USA
srivatsan@g.harvard.edu

Srinivas Sridharan

NVIDIA
Santa Clara, CA, USA
srisridharan@nvidia.com

Amir Yazdanbakhsh†

Google DeepMind
Mountain View, CA, USA
ayazdan@google.com

Tushar Krishna†

Georgia Institute of Technology
Atlanta, GA, USA
tushar@ece.gatech.edu

Vijay Janapa Reddi†

Harvard University
Cambridge, MA, USA
vj@eecs.harvard.edu

Abstract

Large-scale machine learning models necessitate distributed systems, posing significant design challenges due to the large parameter space across distinct design stacks. Existing studies often focus on optimizing individual system aspects in isolation. This work challenges this limitation and introduces **COSMIC**, a full-stack distributed machine learning systems environment enabling end-to-end simulation and agent-based design space exploration. To facilitate efficient exploration and optimization across the entire stack, we introduce **Parameter Set Architecture**—an abstraction concept analogous to the instruction set architecture—abstracting away configuration complexities of agent-based search methods. Case studies demonstrate COSMIC’s ability to consolidate parameters across multiple layers of design abstraction, discovering eight non-obvious high-performance system configurations across four transformer-based models with up to 175 billion parameters. By optimizing across the stack, COSMIC full-stack optimization delivers $1.50\text{--}48.41\times$ higher performance compared to the isolated single-stack optimization.

Keywords

parameter set architecture, cosmic, distributed machine learning system, agent-based full-stack optimization

1 Introduction

Recent advances in large-scale distributed machine learning (ML) models have revolutionized numerous applications, from protein folding [19] and mathematical reasoning [53] to performance optimization [12, 47] and high-quality video generation [13, 48]. These capabilities are enabled by increasingly sophisticated transformer-based models such as Google Gemini [40], OpenAI ChatGPT [38] and Sora [29], Claude Sonnet [3], and Meta LLaMA [31]. The scale of these models continues to grow exponentially, with parameters now reaching from billions to trillions [8, 41], and no indication

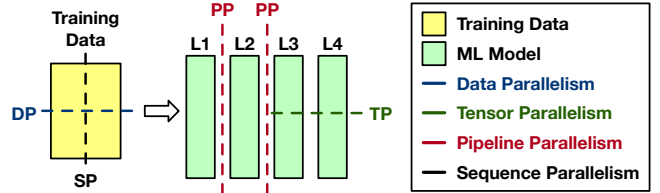


Figure 1: Common parallelization strategies common in distributed ML: DP, SP, PP, and TP.

of slowing down—their memory requirements now far exceed the tens of GBs of storage available on a single compute node [33].

Executing these models poses significant resource and environmental challenges due to their unprecedented computational demands. The scale of the problem is stark: training GPT-3 on a single NVIDIA V100 graphics processing unit (GPU) would require an impractical 355 years [26]. Consequently, *large-scale distributed ML* has become essential [25, 27, 46, 49], requiring sophisticated strategies to partition and distribute models and data across thousands of neural processing units (NPUs), including tensor processing units (TPUs) [17, 18] and GPUs [2, 33, 34]. The complexity of these distributed ML systems spans multiple interconnected design layers—*Workload*, *Collective*, *Network*, and *Compute*—each incorporating critical elements such as model partitioning methods [46, 55], synchronization schedules [43], communication protocols [52], and foundational compute technologies [17, 18]. Given the aggregate influence of these components on ML execution performance, a co-design approach is essential.

Optimizing large-scale distributed ML systems involves three main challenges: (i) **parameterizing every component within the distributed ML stack**, (ii) **creating comprehensive simulation capabilities to predict the cost of executing multi-node deep learning (DL) models**, and (iii) **productively and efficiently exploring the extensive parameter space for customization**. The scale of this co-design space is staggering—even with just four parallelization dimensions (DP, SP, TP, PP), each ranging from 1 to 1024, there are

*These authors contributed equally to this research.

†Equal advising.

286 possible combinations. This complexity is further compounded by additional parameters such as **link bandwidth allocation**, **chunks per collective**, and **multi-dimensional collective optimization** [7]. Furthermore, combined with collective scheduling options, choices for communication algorithms, topology options, and possible NPU distributions, the design space explodes to approximately 7.69×10^{13} potential configurations (see Table 1). Even with an optimistic simulation time of one second per design point, an exhaustive search would require an impractical 2.44×10^6 years, **making traditional search methods infeasible** as systems scale towards larger models, higher NPU counts, and more complex topologies.

Prior design space exploration (DSE) methods have attempted to manage this complexity by focusing on isolated components of the system stack [20, 28, 30, 44, 56, 57, 59, 60]. Although some approaches have employed agent-based methods to optimize parameters within specific layers [20, 30], this isolated optimization strategy can lead to several limitations. First, optimizing individual components in isolation risks overfitting to local optima, potentially missing critical interactions between different layers of the stack. Second, the lack of a unified abstraction framework makes it challenging for domain experts to effectively utilize these optimization methods across different system configurations. Third, focusing on subset optimizations limits the agent’s ability to discover non-obvious solutions that might emerge from cross-layer interactions. These limitations become particularly acute as model sizes and system complexity continue to grow, necessitating a more comprehensive approach to system optimization.

To address these challenges, we propose a two-part solution: (i) Parameter Set Architecture (PSA), an abstraction layer for defining and managing the complex parameter space of distributed ML systems, and (ii) COSMIC¹, a framework that leverages PSA to enable efficient artificial intelligence (AI) agent-based exploration of full-stack distributed ML platform designs.

The PSA is analogous to how an instruction set architecture (ISA) defines the interface between software and hardware. It defines the interaction between search agents and the underlying system using a standard schema-based approach, enabling domain experts to specify searchable parameters, valid ranges, and design constraints without requiring direct agent configuration. The PSA scheduler *automatically* establishes the abstraction layer between agents and the design space, facilitating efficient exploration of configurations across all system layers.

Leveraging PSA, we introduce COSMIC, a framework that conjoins full-stack large-scale distributed ML with agent-based search capabilities. COSMIC combines ASTRA-sim [58]’s detailed distributed ML simulation capabilities with ArchGym [23]’s flexible ML-agent framework, creating a comprehensive environment for exploring distributed ML system designs. By leveraging PSA’s abstraction layer, COSMIC enables automatic configuration of both the simulation environment and search agents, allowing efficient exploration of designs spanning from workload parallelization strategies to network topology choices. The framework supports multiple ML-based search algorithms—including Bayesian optimization (BO), ant colony optimization (ACO), and genetic algorithms (GA)—each bringing unique exploration strategies to navigate the vast design

space. Using this approach in our evaluation, we discover several distinct large-scale distributed system configurations, each exhibiting similar performance characteristics while offering different trade-offs in system design. This diversity of high-performing configurations demonstrates the effectiveness of our approach in navigating the complex co-design space and identifying non-obvious solutions that would be difficult to discover through traditional optimization approaches.

This work makes the following contributions:

- We introduce Parameter Set Architecture (PSA)—akin to the role of ISA in delineating the interface between software and hardware—*PSA delineates the interface between search agents and the underlying system*. It offers a schema for defining searchable parameters and their ranges along with design constraints. This approach shields domain experts from the complexities of agent setup and configuration while enabling seamless exploration of design spaces comprising millions of parameters.
- We introduce COSMIC—a full-stack large-scale distributed ML simulation and agent-based search mechanism combined, built upon PSA. COSMIC can identify distinct system configurations, including workload parallelization strategies, collective algorithms, and network topologies, with similar performance traits, providing system designers with multiple viable implementation options.
- We evaluate four transformer-based models (up to 175B parameters) and perform more than six million steps across four search agents. Our results demonstrate that diversifying parameters across the entire design stack of distributed ML systems outperforms approaches that merely increase the design space of isolated components. Specifically, we show that full-stack optimization enabled by COSMIC delivers $1.50\text{--}48.41 \times$ higher performance compared to the isolated single-stack optimization.

2 Background

Distributed ML of large models involves complex interactions across multiple stacks. To understand the design space and optimization challenges addressed by our work, we first review four key aspects of distributed ML systems: (i) parallelization strategies that enable model and data distribution, (ii) collective communication patterns that facilitate data synchronization, (iii) network topologies that affect communication performance, and (iv) compute devices that execute the ML operations. Each of these components presents unique trade-offs and constraints that must be considered in full-stack co-design optimization.

2.1 Parallelization Strategies

We describe four key workload parallelization approaches in distributed ML, which are summarized in Figure 1.

Data Parallelism (DP) divides the input data set into smaller batches and processes them concurrently across multiple NPUs, increasing the throughput [55]. However, DP does not reduce the memory footprint requirement of each compute device.

Sequence Parallelism (SP) splits input activation across the sequence dimension, particularly for large language models (LLMs) [37]. SP can yield better throughput for operations like dropout or normalization. However, SP does not reduce NPU memory footprint.

¹Cross-stack Optimization and System Modeling with Intelligent Co-design

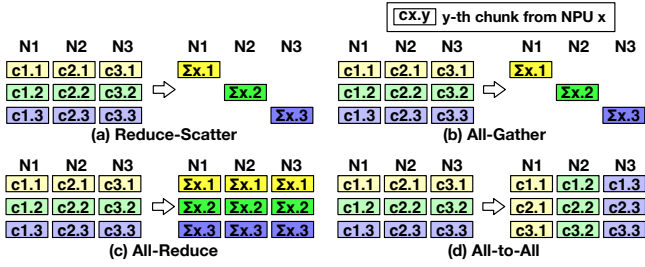


Figure 2: Common collective communication patterns incurred in distributed ML for NPU synchronization.

Tensor Parallelism (TP) vertically splits the deep neural network (DNN) model, distributing each layer across multiple NPUs [46]. TP can help reduce the memory requirement of each NPU, but it introduces frequent and costly communication overhead.

Pipeline Parallelism (PP) distributes the DNN across the dimension of the ML layer [14, 15]. Each NPU processes a subset of layers and forwards the activation. PP reduces memory usage per NPU, but efficient pipeline orchestration is necessary to prevent bottlenecks.

2.2 Collective Communications

Due to the distributed nature of models, NPUs must periodically synchronize data, such as activations or weight gradients. This process relies on *collective communications*, specialized synchronization patterns that manage data flow across the network, processed in fine-grained *chunks* [52]. Figure 2 summarizes commonly used collective patterns in distributed ML execution.

Reduce-Scatter reduces corresponding chunks from every NPU and then distributes the result.

All-Gather broadcasts a chunk of each NPU to all other NPUs. This makes all chunks accessible to the entire cluster.

All-Reduce is the most commonly used collective [22]. All chunks are reduced and broadcasted, which is essential for aggregating weight gradients to synchronize ML models.

All-to-All patterns occur when each NPU generates and transfers dedicated chunks for all other NPUs, such as gating functions in Mixture-of-Experts (MoE) models [45].

Several *collective communication algorithms* define the traffic patterns to implement and execute collective communication patterns. Collective communication libraries (CCLs), such as NCCL [35], implement multiple collective algorithms, including Ring (RI) [6], Direct (DI) [42], Recursive Halving-Doubling (RHD) [52], and Double Binary Tree (DBT) [16].

2.3 Network Topology

The network fabric processes message transfers in distributed DL systems. Prior work in network technology, such as high-speed interconnects like InfiniBand [50] improves performance and efficiency. Different network technologies have distinct performance and cost implications, leading to multiple network fabrics in state-of-the-art clusters. For example, Google Cloud TPU [17] uses a 3D Torus topology with electrical and photonic networks, whereas NVIDIA uses multi-dimensional switches through NVLink [36], NVSwitch [36], and InfiniBand [50]. Cerebras [5] and Tesla [51]

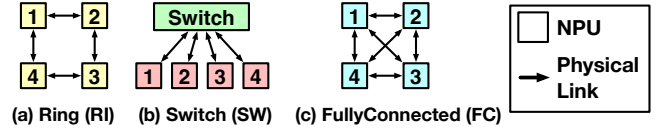


Figure 3: Network topology building blocks we considered to construct multi-dimensional topologies.

clusters use wafer-scale engines with on-chip networks and electrical scale-out capabilities. We abstract these core topologies using the multi-dimensional network representation from [58]. Ring (RI), Switch (SW), and FullyConnected (FC), the core network building blocks in this notation, are summarized in Figure 3. Multi-dimensional network fabrics are represented by stacking these blocks, with link bandwidth and latency information for each dimension (e.g., 3D Torus can be captured as [RI, RI, RI]).

2.4 Compute Devices

Compute devices refer to the components that run all compute operators of ML workloads. GPUs are the most prevalent compute accelerators in ML systems, with state-of-the-art examples including A100 [33] and H100 [34] from NVIDIA, and MI300 [2] from AMD. For additional energy efficiency, there also exist custom accelerators such as TPU [18] from Google, Trainium [1] from Amazon, CS2 [5] from Cerebras. In this work, we refer to the compute devices as NPUs.

In our case studies, we model the compute device by three parameters: *peak-perf*, *local-mem-bw*, and *memory-capacity*. With the first two parameters, we can create a simple rooftop model to get the runtime of each operator. With the last parameter, we can apply the constraint of the memory capacity, which will affect the design space for parallelization strategies.

3 Motivation for Full-Stack Co-Design

The rapid growth of LLMs requires efficient distributed ML systems. Optimizing these systems involves a complex co-design space that spans workload, system, network, and compute characteristics.

3.1 Opportunities for Full-stack Optimization

Traditional parameter tuning approaches in distributed ML systems often focus on optimizing individual components in isolation [20, 30, 56]. This isolated optimization fails to capture the complex interactions between different layers of the system stack, potentially missing significant performance opportunities.

Our preliminary analysis, summarized in Figure 4, demonstrates the importance of considering these cross-layer interactions. For instance, Figure 4(a) fixes the physical cluster and observes how the workload parallelization strategy can impact the GPT3-175B training latency. Even when considering only workload-level parameters, the latency can vary up to 64.5 \times .

This significant improvement from tuning a single design stack suggests substantial untapped potential in the broader design space. When we expand our search to simultaneously optimize the entire design stack, the performance improvements amplify dramatically. As shown in Figure 4(d), the co-designed cluster can achieve latency reductions of up to 103 \times . This multiplicative effect reveals the presence of powerful cross-layer optimization opportunities.

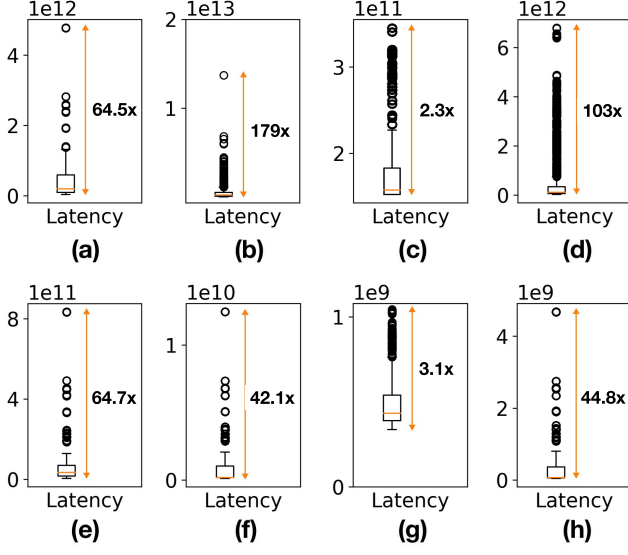


Figure 4: (a) shows latency spread for training GPT3-175B just varying the workload parameters (i.e., workload-only search for System 2, see Section 5). Notably, the parallelization optimal for the target cluster achieved 64.5 \times better performance compared to the worst case, highlighting co-optimization potential. (b)–(d) shows workload+network, workload+collective, and full-stack optimization results for GPT3-175B. (e) latency spread for workload-only DSE for GPT3-13B, (f) workload-only DSE for ViT-Large, (g) full-stack DSE for ViT-Large, and (h) full-stack DSE for ViT-Base.

These results highlight two insights: (i) neglecting parameter optimization across layers can potentially lead to significant performance penalties in distributed ML systems. Local optima at individual layers may conflict with global system efficiency. Also, (ii) the vast co-design space with its intricate dependencies necessitates intelligent search agents capable of efficiently identifying better configurations across workload, system, and network components.

3.2 Challenges of Tackling the Co-design Space

While full-stack optimization is important in efficient distributed ML systems, their design space suffers from a combinatorial explosion, making exhaustive search impractical. Consider a system with a 4D network topology and 1,024 NPUs. Different design knobs and a number of potential design points are summarized in Table 1.

Parallelization dimensions (DP, PP, SP, TP), each ranging between (1, 1, 1, 1024), (1, 1, 2, 512), \dots , (1024, 1, 1, 1), already creates 286 potential options. This can be multiplied with two collective scheduling policy choices, 256 ($=4^4$) 4D collective communication algorithms, 81 ($=3^4$) physical network topology options, etc., further exploding the design space. The total combination of the co-design space for this 1,024 NPU cluster results in about 7.69×10^{13} potential designs. Even with an optimistic simulation time of one second per design point, an exhaustive search would take an impractical 2.44×10^6 years. As systems scale towards larger models, higher NPU counts, and more complex topologies, this combinatorial explosion renders naive search methods infeasible.

Table 1: PSA schema to capture distributed ML design space. The number of design points are retrieved by assuming a 4D network with 1,024 NPUs (see Section 3.2).

Workload Knob	Value Range	#Points
DP	{1, 2, 4, 8, \dots , 512, 1024}	.
PP	{1, 2, 4, 8, \dots , 512, 1024}	.
SP	{1, 2, 4, 8, \dots , 512, 1024}	286
Weight Sharded	{0, 1}	2
Collective Knob	Value Range	#Points
Scheduling Policy	{LIFO, FIFO}	2
Collective Algorithm	MultiDim {Ring, Direct, RHD, DBT}	256
Chunks per Collective	{1, 2, 3, 4, \dots , 32}	32
Multi-dim Collective	{Baseline, BlueConnect}	2
Network Knob	Value Range	#Points
Topology	MultiDim {Ring, Switch, FC}	81
NPUs per Dim	MultiDim {4, 8, 16}	81
Bandwidth per Dim	MultiDim {100, 200, 300, 400, 500}	625
Total #Points		7.69×10^{13}
Constraints		
product (DP, SP, PP) \leq (Number of NPUs) = 1,024		
product (NPUs per Dim) = (Number of NPUs) = 1,024		

3.3 Toward an ML-Driven Approach

The exponential growth of distributed ML platform design spaces has led to the increasing adoption of ML-driven search agents for system optimization. These agents—from Bayesian optimization (BO) [24, 32] and genetic algorithms (GA) [21] to random walk (RW) exploration [39]—have demonstrated remarkable success in navigating high-dimensional parameter spaces [11, 20]. Their diverse strengths complement different aspects of the search problem: BO efficiently finds global optima with limited samples, GA excels at discrete parameter spaces, and even simple RW approaches provide valuable baseline exploration capabilities.

However, leveraging these ML agents effectively creates a significant knowledge barrier. Current optimization frameworks like ArchGym [23] force an unnecessary coupling between domain expertise and ML knowledge: system designers must understand the internals of the ML agent to define observation and action spaces, while ML experts need deep domain knowledge to properly configure the search space. This coupling becomes problematic when parameters require modification across different layers of the distributed ML platform stack, when new design spaces or system configurations need to be explored, or when different ML agents need to be evaluated on the same design space. This intricate interdependency between system design knowledge and ML expertise creates a bottleneck in the optimization process, highlighting the need for a systematic decoupling approach.

4 COSMIC: Parameter-Based Approach for DSE

To address the myriad challenges, we introduce COSMIC, a framework that implements Parameter Set Architecture (PSA) to enable systematic exploration of distributed ML design spaces. COSMIC creates a clean separation between design parameters and search mechanics through two key capabilities: (i) a structured parameter definition schema and (ii) an automated scheduler that configures ML agents. This approach eliminates the need for domain experts to understand the internals of ML agents while allowing the ML agents to efficiently explore the hardware design spaces.

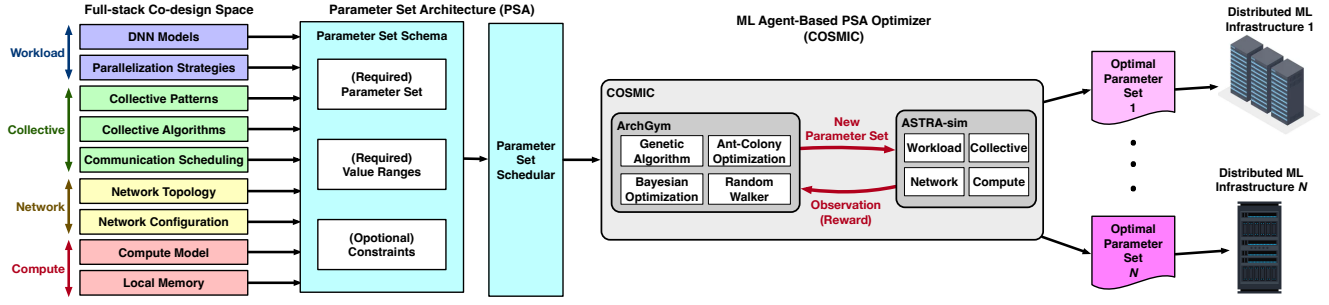


Figure 5: Summary of (i) PSA to capture the full-stack distributed ML design space and (ii) ML-based PSA optimization framework (CosMIC) to design new distributed ML infrastructures.

4.1 Design Principles

COSMIC’s design is guided by three fundamental principles that together enable efficient exploration of distributed ML design spaces: **Separation of Concerns**. The primary goal of COSMIC is to establish clear boundaries between domain knowledge and ML expertise. Domain experts should be able to express complex design spaces and constraints using familiar terminology and concepts, without having to understand the internals of the ML agent. Conversely, ML agents should be able to efficiently explore these spaces without requiring deep knowledge of system design. This separation enables each group to focus on their core competencies while still working or collaborating effectively together.

Automated Configuration. COSMIC eliminates manual agent configuration through automation. Rather than requiring experts to manually define observation spaces, action spaces, and reward functions, COSMIC translates domain-specific parameter definitions into ML-agent configurations. This automation not only reduces the potential for errors but also enables rapid experimentation with different ML agents on the same design space, as the configuration process is handled systematically rather than manually.

Flexible Expression. The framework must accommodate the complexity of distributed ML systems while maintaining usability. This means supporting simple single-stack optimizations (like parallelization strategies) and complex multistack design spaces (involving workload, collective, and network parameters). Moreover, we must capture real-world constraints and parameter dependencies without sacrificing the clarity of the parameter specification process. This flexibility ensures that COSMIC can address both current optimization needs and future expansions of the design space.

These principles work together to create an abstraction layer that makes full-stack optimization more accessible while maintaining the power and flexibility needed for complex distributed ML systems, whose schema we discuss next.

4.2 Parameter Set Schema

COSMIC relies on its schema definition, which provides a systematic way to express and constrain the design space. The schema serves as a contract between domain experts and ML agents, ensuring that all explored configurations are valid and meaningful. To achieve this, the schema is structured around three components:

Parameter Set. This component defines the specific parameters to be explored, encompassing both single-layer and multi-stack design spaces. These parameters span workload characteristics (e.g., parallelization strategies), collective configurations (e.g., collective algorithms), and network specifications. Each parameter represents a distinct dimension in the design space that can be tuned to optimize the system performance.

Value Range. To ensure practical exploration, each parameter is associated with a defined target range of valid values, carefully selected by experts in the domain. These ranges serve dual purposes: (i) they control the COSMIC scheduler’s configuration space, and (ii) delimit the search boundaries for ML agents. This structured approach prevents agents from exploring invalid or impractical configurations while maintaining flexibility within meaningful bounds.

Constraints. Real-world systems rarely have independent parameters. The schema, therefore, allows domain experts to specify cross-parameter dependencies using additional constraints. These constraints capture relationships between parameters, ensuring that the generated configurations respect the requirements at the system level and technological limitations. Although this increases complexity, it is crucial to generate viable system configurations.

This schema definition allows for flexible and expressive definitions of the design space, capturing both individual parameter ranges and inter-parameter dependencies. The workload stack captures the number of target NPUs in a system and their potential parallelization strategies. The collective stack captures collective-level communication configurations, such as collective scheduling policy, collective algorithms, or whether to enable BlueConnect [7] multi-dimensional optimization. The network stack encapsulates the information about the topology and associated network bandwidth. Cross-stack correlation information is also being captured through constraints. For example, the parallelization strategy is limited by the number of physical NPUs in the cluster.

4.3 Parameter Set Scheduler

Parameter Set Scheduler (PSS) is there for automating the configuration of both search environments and agents. Adapting search mechanisms to changes in parameter sets traditionally requires manual adjustments to both the environment and the agent’s action space. This manual configuration exposes the inherent complexity of heavy domain knowledge, hindering efficient exploration. The PSS overcomes this challenge by abstracting away this complexity,

enabling seamless integration and execution of search agents for design exploration. Below we describe the main design components. **Environment Side Configuration.** The search environment encapsulates the core cost model (e.g., simulators). It receives design parameters as input and estimates desired performance metrics like latency or communication overhead. For efficient interaction with search agents, the environment must clearly define (i) action space, (ii) observation space, and (iii) constraints to prevent ineffectual simulations with invalid parameter combinations. The PSS automates this process by defining the action and observation spaces and incorporating constraints based on the parameter set schema. **Agent Side Configuration.** The PSS configures various aspects of the search agent to ensure efficient DSE. For effective interaction with the environment interface, the agent must clearly define: (i) the agent’s search space, (ii) the type of each parameter (integer, categorical, boolean, float, etc.), (iii) the step size for each parameter as defined in the parameter schema, and (iv) the configuration of reward function parameters. The PSS automates this process, allowing search agents to effectively navigate the design space with well-defined parameters and rewards.

4.4 CosmIC Implementation

To prove and show the effectiveness of our parameter-centric approach, we implement CosmIC, a framework that enables ML-driven full-stack optimization of distributed ML systems. CosmIC integrates two key components: ASTRA-sim’s detailed, end-to-end distributed ML simulation capabilities, and ArchGym’s flexible ML-agent search mechanism. We describe how our parameter-centric approach enables this integration and discuss the key components.

ASTRA-sim is the simulation engine for CosmIC. As an open-source, full-stack distributed ML training simulator, it models intricate interactions across workload characteristics, system parameters, network topology, and compute resources. ASTRA-sim has been cross-validated against real system measurements, ensuring accurate performance analysis of distributed ML.

ArchGym serves as the ML exploration backbone of CosmIC. It streamlines ML-driven architecture exploration by providing a standardized interface between search agents and architectural simulators through the OpenAI Gym library. This standardization eliminates the need for manual agent configuration, allowing pre-configured ML agents to efficiently explore parameter spaces.

Our parameter-centric abstraction enables seamless integration of these components through a well-defined agent-environment interaction loop. At the heart of this integration is the PSS, which translates domain-specific parameter definitions into action spaces that ML agents can understand and explore. When an agent selects a particular system configuration, it submits this action to the ASTRA-sim simulation backend. ASTRA-sim then performs a detailed simulation of the distributed ML system with the specified parameters, generating performance metrics and corresponding reward signals. These results are processed by ArchGym’s environment layer, which uses this feedback to guide agents toward increasingly better configurations. This continuous feedback loop, mediated by our parameter-centric abstraction, enables efficient exploration of the design space while maintaining separation between domain expertise and ML implementation details. This PSA-enabled

Table 2: Target Workloads.

*: To expedite runtime, we simulate 4 layers per model and scale memory capacities to correctly reflect memory constraints. In post-processing, we re-scale latency and memory to reflect the whole model performance.

Parameters	GPT3-175B	GPT3-13B	ViT-Base	ViT-Large
Number Layers*	96	40	12	24
Embedding Dimension	12288	5140	768	1024
FFN Dimension	49152	20560	3072	4096
Sequence Length	2048	2048	256	256
Number Heads	96	40	12	16

Table 3: System configurations used for evaluation.

Knobs	System 1	System 2	System 3
Collective Knob			
Collective Algorithm	[RI, RI, RI, RHD]	[RI, DI, RI, RHD]	[DI, RHD, RI, RI]
Network Knob			
Topology	[RI, RI, RI, SW]	[RI, FC, RI, SW]	[FC, SW, RI, RI]
NPU’s per Dim	[4, 4, 4, 8]	[4, 8, 4, 8]	[8, 16, 4, 4]
Bandwidth per Dim	[200, 200, 200, 50]	[375, 175, 150, 100]	[900, 100, 50, 12.5]
Compute Knob			
Compute Performance	459	10	900
Local Mem BW	2765	50	3000

integration significantly simplifies the exploration of full-stack distributed ML design spaces by abstracting away the complexity of component interaction while maintaining the flexibility to evaluate diverse system configurations.

ASTRA-sim receives a workload trace to be simulated. Distinct workload parallelization yields a completely different workload trace. Consequently, to enable the exploration of workload knobs as defined in the PSA, we equipped the CosmIC framework with a Workload Trace Generator (WTG). The WTG defines multiple trace templates, which capture and represent the ML workload architecture in terms of atomic operators. For example, these templates show structures like multi-head attention mechanisms. Compared to the complete compute trace, such a template maintains flexibility as (i) it is not represented in exact numbers and instead uses numeric symbols, e.g., {B, S, D, H}, and (ii) the partitioning of tensors is also symbolically captured using the workload knobs, e.g., {tp, dp}. When the PSA search knobs are provided by the PSS, the WTG translates the trace template into an actual trace to be simulated by ASTRA-sim. To retrieve the trace, it first substitutes the numeric symbols in the template using the actual PSA knobs to get actual tensor shapes and sizes. Then, the generator analyzes all tensors, especially their shape and producer/consumer NPU information, and injects collective communications whenever necessary.

5 Experimental Setup

This section outlines the target workloads, baseline system configurations, and experimental setup used to demonstrate the advantages of PSA and CosmIC.

5.1 Target Workloads and Systems

We leverage four LLMs of varying scales to evaluate our approach: GPT3-13B, GPT3-175B [4], ViT-Base, and ViT-Large [10]. Details of each model are summarized in Table 2.

Table 4: Target PSA used for evaluations in this work.

Workload Knob	Value Range
DP	{1, 2, 4, 8, \dots , 1024, 2048}
PP	{1, 2, 4}
SP	{1, 2, 4, 8, \dots , 1024, 2048}
Weight Sharded	{0, 1}
Collective Knob	Value Range
Scheduling Policy	{LIFO, FIFO}
Collective Algorithm	MultiDim {Ring, Direct, RHD, DBT}
Chunks per Collective	{2, 4, 8, 16}
Multi-dim Collective	{Baseline, BlueConnect}
Network Knob	Value Range
Topology	MultiDim {Ring, Switch, FC}
NPU's per Dim	MultiDim {4, 8, 16}
Bandwidth per Dim	MultiDim {50, 100, 150, \dots , 450, 500}
Constraints	
product (DP, SP, PP) \leq (Number of NPUs)	
product (NPUs per Dim) = (Number of NPUs)	

For baseline targets, we evaluate three systems differing in the number of NPUs and corresponding real-world configurations:

- **System 1:** Represents a cluster of 512 Google TPuv5p devices [54].
- **System 2:** Based on a 4D network topology comprising 1,024 NPUs, as described in [43].
- **System 3:** Serves as a proxy for a 2,048-NPU system using NVIDIA H100 GPUs [34].

Detailed specifications for each target system are provided in [Table 3](#).

5.2 PSA Parameters

Each experiment selects a specific target system and employs CosMIC, guided by PSA, to optimize either a fixed stack for baseline comparisons or a full-stack configuration through automated search. As discussed in [Section 4](#), the optimization performed by CosMIC is enabled by the abstraction layer provided by PSA. [Table 4](#) summarizes the PSA configurations used in our evaluation, specifying the target values for each distributed ML design parameter exposed by PSA during system optimization.

5.3 ML Search Agents

During ML-based optimization, CosMIC utilized one of four ML agents. Thanks to the abstraction provided by PSA, the implementation of distributed ML modeling is entirely decoupled from agent-based search. As a result, any reinforcement learning (RL) algorithm can be integrated with CosMIC².

Random Walker (RW) [39]. For RW, we vary the population size, i.e., the number of agents.

Genetic Algorithm (GA) [21]. For GA, we tune both the population size and the mutation probability.

Ant Colony Optimization (ACO) [9]. For ACO, we adjust the number of ants (agents), the greediness factor, and the pheromone evaporation rate.

Bayesian Optimization (BO) [32]. For BO, we randomize the surrogate model by varying the random seed of the underlying Gaussian process.

²We did not evaluate RL methods in this work due to their significantly higher simulation time per sample (approximately 20–30 minutes).

5.4 Optimization Objectives

CosMIC is designed to maximize ML performance, specifically by minimizing total ML runtime. However, unconstrained exploration of the network configuration space may lead CosMIC to select overly resource-intensive setups—for example, assigning the maximum available bandwidth to each network dimension. To avoid this, we introduce additional constraints to regulate the network search space.

Runtime per BW/NPU. In this setup, CosMIC aims to maximize ML performance while limiting the bandwidth allocated per NPU. To achieve this, CosMIC minimizes the product of ML execution latency and the total network bandwidth consumed per NPU. A minus-one offset is applied to prevent divide-by-zero errors in cases where invalid configurations produce a latency or bandwidth of zero.

$$\text{reward}_{(\text{Perf per BW/NPU})} = \frac{1}{\sqrt{(\text{Latency} \times \sum (\text{BW per Dim}) - 1)^2}}$$

Runtime per Network Cost. As an alternative regulation metric, we assess ML performance in relation to network dollar cost. We adopt the cost model proposed in [59], and define the reward function as:

$$\text{reward}_{(\text{Perf per Network Cost})} = \frac{1}{\sqrt{(\text{Latency} \times \text{Network Cost} - 1)^2}}$$

Additionally, any parallelization strategy resulting in a memory footprint exceeding 24 GB per NPU is considered invalid and discarded.

6 Results

This section presents experimental results and evaluates the impact of co-optimizing different design stacks in distributed ML systems. We also examine the effect of using different ML agents.

6.1 Full-Stack Optimizations

We first demonstrate the benefits of full-stack optimization using CosMIC. Such end-to-end optimization is made possible by the abstraction provided by PSA, as outlined in [Table 4](#), which enables seamless integration with ML-based DSE frameworks.

We employed CosMIC to perform DSE to optimize the training of the GPT3-175B workload. Specifically, CosMIC tunes the PSA parameters to maximize the reward functions defined in [Section 5.4](#).

To assess the contributions of each layer, we restricted CosMIC to operate on individual stacks:

- **Workload-only search:** Optimizes parallelization parameters (DP, PP, SP) and weight sharding flags, analogous to approaches like FlexFlow [30] and UniAP [28].
- **Collective-only search:** Tunes collective algorithms, serving as a proxy for communication-optimization works.
- **Network-only search:** Adjusts network topology and resource allocation, reflecting system-level optimization efforts.
- **Full-stack search:** Optimizes all parameters exposed by PSA.

CosMIC was evaluated on System 1 (512 NPUs) and System 2 (1,024 NPUs). Results are shown in [Figure 6](#) and [Figure 7](#).

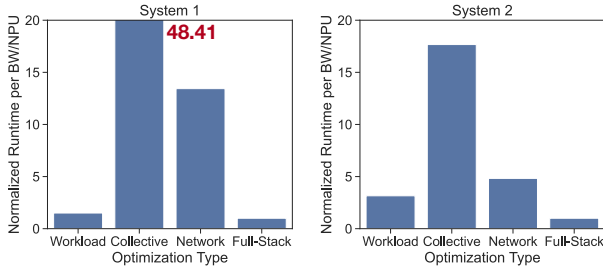


Figure 6: ML runtime per BW/NPU for the GPT3-175B workload. The base systems were System 1 and System 2. Each bar represents a scenario where COSMIC is restricted to workload-only, collective-only, network-only, or full-stack optimization. Results are normalized to the full-stack outcome.

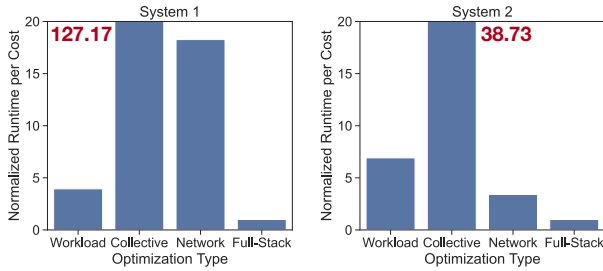


Figure 7: Normalized ML runtime per network dollar cost for GPT3-175B. Similar to Figure 6, values are normalized to the corresponding full-stack search result.

Figure 6 shows that full-stack optimization consistently yields the best performance per BW/NPU across both systems. This highlights the importance of jointly optimizing workloads, collectives, and network components. For System 1 and System 2, the full-stack optimization yielded performance improvements of 1.50–48.41× and 3.15–17.67×, respectively, over single-stack baselines.

Among the single-stack variants:

- Collective-only optimization provided the least gain, as it only improves communication efficiency.
- Network-only optimization achieved more benefits by improving both communication and BW/NPU ratios.
- Workload-only optimization had the highest single-stack gains, as it can influence communication patterns and compute communication overlap.

Figure 7 reports performance per dollar cost. Here, full-stack optimization led to even greater improvements: 3.94–127.17× for System 1 and 3.40–38.73× for System 2, relative to single-stack baselines. Interestingly, for System 2, network-only optimization outperformed workload-only optimization due to a significant impact on cost, despite smaller speedups. Table 5 summarizes the full-stack DSE results for System 2. Depending on the applied optimization constraint—BW/NPU or network dollar cost—the ML agent produces entirely different network configurations. These network choices, in turn, influence other design stacks, including workload parallelization strategies and selected collective algorithms. This highlights the tightly coupled nature of the ML system

Table 5: System configurations used for evaluation.

Target	Perf per BW/NPU	Perf per Network Cost
Workload Knob		
DP	64	128
PP	1	1
SP	4	4
Weight Sharded	1	1
Collective Knob		
Scheduling Policy	LIFO	LIFO
Collective Algorithm	[RI, RHD, DBT, DBT]	[RHD, RI, RI, RHD]
Chunks per Collective	4	16
Multi-dim Collective	Baseline	Baseline
Network Knob		
Topology	[RI, RI, RI, SW]	[SW, FC, FC, FC]
NPUs per Dim	[4, 4, 4, 16]	[4, 8, 4, 8]
Bandwidth per Dim	[50, 50, 50, 50]	[50, 50, 50, 50]

design space and reinforces the importance of performing full-stack optimizations.

In summary, full-stack system design consistently outperformed partial-stack approaches in both performance-per-BW/NPU and performance-per-cost metrics. Across all experiments, full-stack optimization delivered at least 50–240% improvement, underscoring the value of holistic system design made possible by PSA and its integration within COSMIC.

6.2 Scalability Analysis

Next, we demonstrate the scalability of PSA and COSMIC for larger systems and workloads. In this section, we leverage system 3 as the target baseline, which consists of 2,048 NPUs and is 4× and 2× larger in number of NPUs than system 1 and 2 used in Section 6.1, respectively. For this larger target system, we use two distinct workloads with different scales: one set of experiments using a relatively small, ViT-Large model, and the other experiments targeting the larger GPT3-175B. Finally, we also execute DSEs with different global batch sizes, from 1,024 to 16,384 samples, to explore the scalability of COSMIC in terms of batch sizes (thereby increasing computation and communication requirements) as well. Since workload-only optimization overall provided the best performance within the single-stack optimization target, thus we used workload-only optimization as our target baseline to compare full-stack optimizations.

Figure 8 summarizes the experimental results for COSMIC scalability. It’s worth noting that:

- Regardless of the increased system size (2,048 NPUs), the full-stack optimization consistently showed great performance benefits over the workload-only optimization.
- Also, the performance benefit of the full-stack search was consistent regardless of the workload size: ViT-Large or GPT3-175B.
- Finally, for every batch size, full-stack optimization always yielded better performance than workload-only optimization as well.

For ViT-Large, the performance benefit of the full-stack-optimized system over the workload-optimized cluster was 2.30×, 2.38×, 1.71×, 2.31×, 3.75× for global batch sizes 1,024–16,384. Full-stack optimization provided at least 71% performance improvement over the workload-only design. Likely, GPT3-175B workload provided 4.77×,

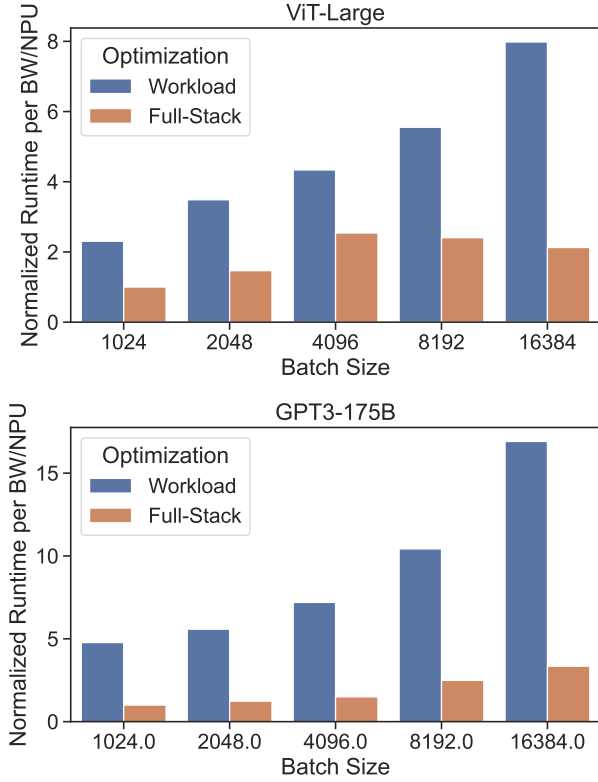


Figure 8: Workload-only vs. full-stack DSE result by CosMIC. The target was system 3 with 2,048 NPUs. Two workloads with different scales, ViT-Large and GPT3-175B were used as the target model. All results were normalized over the corresponding full-stack optimization result with 1,024 batches.

4.52×, 4.81×, 4.19×, and 5.05× over increasing global batch sizes, respectively. The benefit for full-stack optimization was larger for GPT3-175B than ViT-Large, showing at least 4.19× benefit compared to the 1.71×. Larger models, especially over large clusters, show higher resource demands in both compute and communication. Therefore, judicious DSE and optimization are pivotal for bigger models over larger clusters. Furthermore, it denotes more potentials that the full-stack co-design contains over a single-stack optimization, as exemplified by the experimental results in Figure 8. Therefore, we claim that the enabler for full-stack DSE, co-design, and optimization, such as the abstraction of PSA and CosMIC exploration capabilities, will get even more importance as the model and the cluster scales further.

6.3 Co-Design Use Cases

While Section 6.1 and Section 6.2 focused on full-stack optimization, the versatility of PSA enables DSE at arbitrary levels of the ML system stack. To demonstrate this flexibility, we conducted two additional co-design experiments, with results summarized in Table 6.

Experiment 1 targets workload-network co-design while keeping collective algorithms fixed. This experiment jointly optimizes across

Table 6: CosMIC designed parameters for different use cases.

*: **Searching**: Searched parameter; **Fixed**: Fixed value;

	Expr. 1	Expr. 2.1	Expr. 2.2
Observations	Multi-Model	Chat Inference	QA Inference
Network Knobs			
Topology	[RI, FC, RI, FC]	[RI, RI, RI, RI]	[RI, RI, RI, RI]
NPUs-count	[16, 4, 4, 4]	[4, 16, 4, 4]	[16, 4, 4, 4]
Bandwidth per Link	[50, 50, 50, 50]	[50, 50, 50, 50]	[50, 50, 50, 50]
Collective Knobs			
Scheduling Policy	LIFO	LIFO	LIFO
Chunks per Collective	4	2	2
Collective Algorithm	[RI, RI, RI, RI]	[RHD, DBT, DBT, DI]	[DI, DI, DBT, RHD]
Multi-dim Collective	Baseline	BlueConnect	Baseline
Workload Knobs			
Number of NPUs	1024	1024	1024
DP, PP, SP, TP	2, 1, 8, 64	8, 4, 8, 4	8, 4, 8, 4
Weight Sharded	1	1	1

all four workloads, allowing CosMIC to perform DSE across diverse training requirements. Interestingly, when optimizing for an ensemble of models, CosMIC consistently reduces memory footprint by increasing the TP size. To accommodate the enlarged TP groups, CosMIC allocates 64 NPUs along the first two dimensions—matching the TP group size exactly, and illustrating CosMIC’s ability to co-optimize workload and network topology. Since the SP size is 8 and spans the final dimension, CosMIC selected the FC topology to avoid congestion, demonstrating intelligent adaptation to overlapping parallelism strategies.

Experiment 2 explores collective-network co-design while fixing the workload parallelization strategy. This experiment targets inference for GPT3-175B and includes two sub-scenarios: experiment 2.1 (Chat) and experiment 2.2 (question-answer).

As both use cases represent inference tasks, latency-optimized collectives are preferred over bandwidth-optimized ones due to the small message sizes during the decode phase. Accordingly, CosMIC consistently selected latency-efficient collective algorithms—Direct, RHD, and DBT—while avoiding Ring, which is typically bandwidth-optimized. Still, CosMIC opted to split collective operations into two chunks to enable pipelining during larger collectives encountered in the prefill phase of inference.

The results in Table 6 underscore the generality of CosMIC. Thanks to the abstraction provided by PSA, CosMIC supports a range of co-design scenarios—including multi-model training and distinct inference workloads—without requiring any system-specific customization. CosMIC seamlessly adapts to partial-stack co-designs and diverse target objectives, showcasing its versatility in distributed ML system optimization.

6.4 Comparing ML Agents

We now compare the effect of different ML agents when executing full-stack DSE in CosMIC. As described in Section 5.3, CosMIC supports any ML and RL agents without modification, due to the modular abstraction provided by PSA. This section evaluates CosMIC using four distinct ML agents, comparing their final design outputs and convergence behavior.

Figure 9 shows two best-performing system configurations for each agent in experiment 2 of Table 6, targeting GPT3-175B training. Although all agents converged to the same optimal reward, each discovered different design points. Each agent identified multiple configurations yielding equivalent performance by preserving

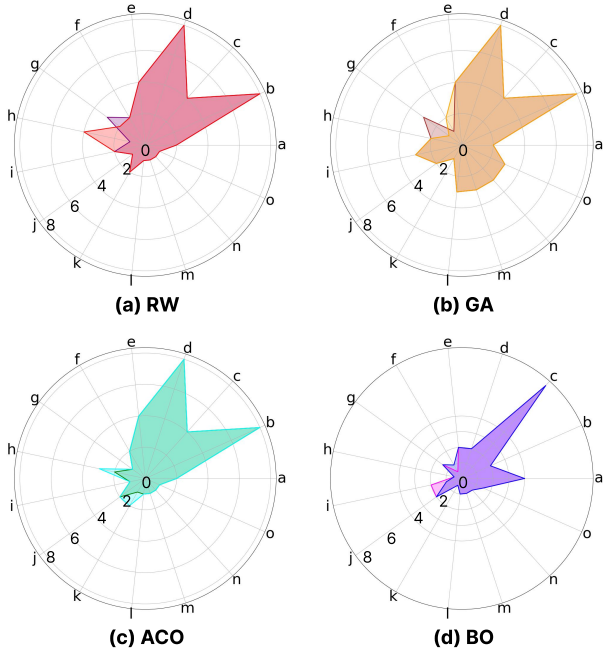


Figure 9: Differing configurations discovered within and across ML agents, all achieving equivalent optimal performance. Parameters are indexed. a): Chunks per Collective; b–e): 4D NPU count; f): Scheduling Policy (1=FIFO, 2=LIFO); g–j): 4D All-Reduce (1=RI, 2=DI, 3=RHD, 4=DBT); k): Multi-dim Collective (1=Baseline, 2=BlueConnect); l–o): 4D Topology (1=RI, 2=FC, 3=SW).

key parameters while exploring variations in others—highlighting the flexibility and redundancy in the design space. This diversity provides system designers with a range of high-performing configurations, enabling trade-off considerations such as cost, complexity, or deployment constraints.

The variation arises from each agent’s unique exploration strategy, initialization behavior, and hyperparameter settings. Notably, each configuration pair per agent in Figure 9 reveals internal consistency in key performance-critical dimensions, with variance in less impactful parameters.

We further compare convergence behaviors in Figure 10, illustrating reward progression over 1,200 optimization steps. The RW agent does not leverage history, resulting in a relatively flat reward curve—achieving high performance purely by chance. In contrast, GA, BO, and ACO exhibit learning behaviors. These agents progressively exploit high-reward regions while avoiding unproductive areas of the search space, as evidenced by upward trends followed by convergence.

All agents ultimately identified multiple optimal full-stack configurations, reaffirming the robustness of CosmIC across algorithmic choices. The number of steps required to reach peak performance was: RW — 652 steps; GA — 440 steps; ACO — 297 steps; BO — 680 steps. These results suggest that more intelligent, history-aware agents can converge more quickly and avoid invalid configurations more effectively.

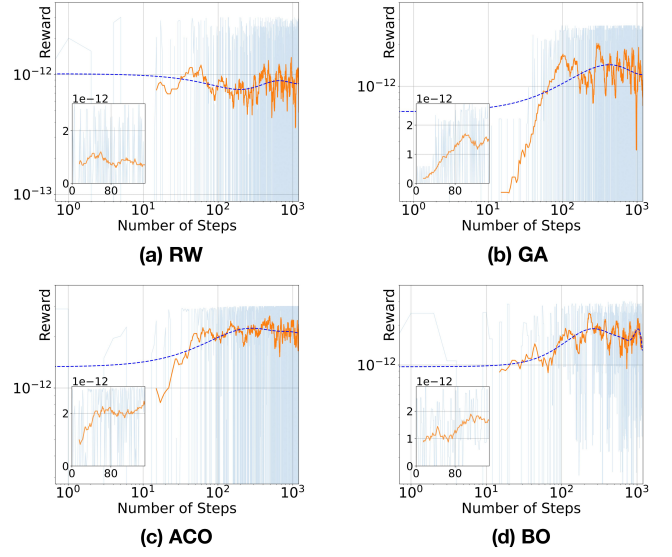


Figure 10: Reward vs. step convergence curves for each ML agent in experiment 4, from 0 to 1,200 steps.

Crucially, CosmIC imposes no constraints on agent selection due to the abstraction provided by PsA. This enables broad applicability to varied optimization goals, use cases, or user preferences. As demonstrated here, CosmIC provides a powerful, agent-agnostic framework for full-stack system co-design at scale.

7 Related Work

Prior works [20, 28, 30, 44, 56, 57, 59, 60] have explored DSE for distributed ML, focusing on optimizing specific components of the system stack. Workload-level optimization targets parallelization strategies given fixed system properties, as seen in FlexFlow [30] (using Markov chain Monte Carlo) and UniAP [28] (using integer quadratic programming). Collective-level DSE focuses on identifying optimal collective algorithms for given network characteristics, exemplified by TACCL [44] and TACOS [57]. Network-level optimization aims to improve collective performance by tuning bandwidth allocation [59] or synthesizing efficient topologies [60]. TopoOpt [56] further explores co-optimizing topology and parallelization using optical network reconfiguration. At the compute level, GAMMA [20] uses genetic algorithms [21] to optimize individual node performance but without considering any of the distributed ML contexts.

In contrast to these isolated approaches, we emphasize the need for full-stack co-design. CosmIC enables holistic system optimization, uncovering bottlenecks across workload, system, network, and compute layers, and leveraging diverse search agents [9, 21, 32, 39] to identify non-obvious design points. This approach allows us to integrate and analyze the impact of optimizations proposed in prior work within a unified framework, providing a better understanding of their effects on end-to-end performance.

8 Conclusion

We introduce CosmIC, a platform combining full-stack large-scale distributed ML system with agent-based search. We propose the

Parameter Set Architecture (PSA) to abstract the interface between search agents and the system, enabling efficient exploration of design spaces with millions of parameters. Our evaluation of transformer-based models demonstrates that COSMIC achieves $1.50\text{--}48.41\times$ higher performance by optimizing the entire design stack.

References

- [1] Amazon. 2020. AWS Trainium. <https://aws.amazon.com/machine-learning/trainium>.
- [2] AMD. 2023. AMD Instinct MI300 Series Accelerators. <https://www.amd.com/en/products/accelerators/instinct/mi300.html>.
- [3] Anthropic. 2024. Introducing the next generation of Claude. <https://www.anthropic.com/news/claude-3-family>.
- [4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.
- [5] Cerebras. 2021. Cerebras Systems: Achieving Industry Best AI Performance Through A Systems Approach. <https://cerebras.net/wp-content/uploads/2021/04/Cerebras-CS-2-Whitepaper.pdf>.
- [6] Ernie Chan, Robert van de Geijn, William Gropp, and Rajeev Thakur. 2006. Collective Communication on Architectures that Support Simultaneous Communication over Multiple Links. In *PPoPP*.
- [7] M. Cho, U. Finkler, M. Serrano, D. Kung, and H. Hunter. 2019. BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy. In *MLSys*.
- [8] Sajal Dash, Isaac Lyngaas, Junqi Yin, Xiao Wang, Romain Egele, Guojing Cong, Feiyi Wang, and Prasanna Balaprakash. 2023. Optimizing Distributed Training on Frontier for Large Language Models. In *arXiv:2312.12705 [cs.DC]*.
- [9] M. Dorigo and G. Di Caro. 1999. Ant Colony Optimization: A New Meta-Heuristic. In *CEC*.
- [10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *arXiv:2010.11929 [cs.CV]*.
- [11] Hadi Esmailzadeh, Soroush Ghodrati, Andrew Kahng, Joon Kyung Kim, Sean Kinzer, Sayak Kundu, Rohan Mahapatra, Susmita Dey Manasi, Sachin Sapatnekar, Zhiang Wang, and Ziqing Zeng. 2024. An Open-Source ML-Based Full-Stack Optimization Framework for Machine Learning Accelerators. *TODAES* (2024).
- [12] Spandan Garg, Roshanak Zilouchian Moghaddam, Colin B. Clement, Neel Sundaresan, and Chen Wu. 2022. DeepPERF: A Deep Learning-Based Approach For Improving Software Performance. In *arXiv:2206.13619 [cs.SE]*.
- [13] Agrim Gupta, Lijun Yu, Kihyuk Sohn, Xiuye Gu, Meera Hahn, Li Fei-Fei, Irfan Essa, Lu Jiang, and José Lezama. 2023. Photorealistic Video Generation with Diffusion Models. In *arXiv:2312.06662 [cs.CV]*.
- [14] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. 2018. PipeDream: Fast and Efficient Pipeline Parallel DNN Training. In *arXiv:1806.03377 [cs.DC]*.
- [15] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *NeurIPS*.
- [16] Sylvain Jeagey. 2019. Massively Scale Your Deep Learning Training with NCCL 2.4. <https://developer.nvidia.com/blog/massively-scale-deep-learning-training-nccl-2-4>.
- [17] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *ISCA*.
- [18] Norman P. Jouppi, Doe Hyun Yoon, George Kurian, Sheng Li, Nishant Patil, James Laudon, Cliff Young, and David Patterson. 2020. A Domain-Specific Supercomputer for Training Deep neural networks. *Commun. ACM* (2020).
- [19] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Židek, Anna Potapenko, et al. 2021. Highly Accurate Protein Structure Prediction with AlphaFold. *Nature* (2021).
- [20] Sheng-Chun Kao and Tushar Krishna. 2020. GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm. In *ICCAD*.
- [21] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. 2021. A Review on Genetic Algorithm: Past, Present, and Future. *Multimedia Tools Appl.* (2021).
- [22] Benjamin Klenk, Nan Jiang, Greg Thorson, and Larry Dennison. 2020. An In-Network Architecture for Accelerating Shared-memory Multiprocessor Collectives. In *ISCA*.
- [23] Srivatsan Krishnan, Amir Yazdanbakhsh, Shvetank Prakash, Jason Jabbour, Ikechukwu Uchendu, Susobhan Ghosh, Behzad Boroujerdian, Daniel Richins, Devashree Tripathy, Aleksandra Faust, and Vijay Janapa Reddi. 2023. ArchGym: An Open-Source Gymnasium for Machine Learning Assisted Architecture Design. In *ISCA*.
- [24] H. J. Kushner. 1964. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering* (1964).
- [25] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *arXiv:2006.16668 [cs.CL]*.
- [26] Chuan Li. 2020. OpenAI's GPT-3 Language Model: A Technical Overview. <https://lambdalabs.com/blog/demystifying-gpt-3>.
- [27] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. In *arXiv:2006.15704 [cs.DC]*.
- [28] Hao Lin, Ke Wu, Jie Li, Jun Li, and Wu-Jun Li. 2024. UniAP: Unifying Inter- and Intra-Layer Automatic Parallelism by Mixed Integer Quadratic Programming. In *arXiv:2307.16375 [cs.LG]*.
- [29] Yixin Liu, Kai Zhang, Yuan Li, Zhiling Yan, Chujie Gao, Ruoxi Chen, Zhengqing Yuan, Yue Huang, Hanchi Sun, Jianfeng Gao, Lifang He, and Lichao Sun. 2024. Sora: A Review on Background, Technology, Limitations, and Opportunities of Large Vision Models. In *arXiv:2402.17177 [cs.CV]*.
- [30] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks. In *HPCA*.
- [31] Meta. 2023. Introducing LLaMA: A Foundational, 65-billion-Parameter Large Language Model. <https://ai.meta.com/blog/large-language-model-llama-meta-ai>.
- [32] J. Mockus. 1975. On Bayesian Methods for Seeking the Extremum. *Optimization Techniques IFIP Technical Conference* (1975).
- [33] NVIDIA. 2020. NVIDIA A100 TENSOR CORE GPU. <https://www.nvidia.com/en-us/data-center/a100>.
- [34] NVIDIA. 2023. NVIDIA H100 Tensor Core GPU. <https://www.nvidia.com/en-us/data-center/h100>.
- [35] NVIDIA. 2024. NVIDIA Collective Communications Library (NCCL). <https://developer.nvidia.com/nccl>.
- [36] NVIDIA. 2024. NVLink and NVLink Switch. <https://www.nvidia.com/en-us/data-center/nvlink/>.
- [37] NVIDIA. 2024. Parallelisms - NVIDIA Docs. <https://docs.nvidia.com/nemo-framework/user-guide/latest/nemotoolkit/features/parallelisms.html>.
- [38] OpenAI. 2022. Introducing ChatGPT. <https://openai.com/blog/chatgpt>.
- [39] KARL PEARSON. 1905. The Problem of the Random Walk. *Nature* (1905).
- [40] Sundar Pichai and Demis Hassabis. 2023. Introducing Gemini: Our Largest and Most Capable AI Model. <https://blog.google/technology/ai/google-gemini-ai>.
- [41] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. In *arXiv:1910.02054 [cs.LG]*.
- [42] Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2020. ASTRA-SIM: Enabling SW/HW Co-Design Exploration for Distributed DL Training Platforms. In *ISPASS*.
- [43] Saeed Rashidi, William Won, Sudarshan Srinivasan, Srinivas Sridharan, and Tushar Krishna. 2022. Themis: A Network Bandwidth-Aware Collective Scheduling Policy for Distributed Training of DL Models. In *ISCA*.
- [44] Aashaka Shah, Vijay Chidambaram, Meghan Cowan, Saeed Maleki, Madan Musuvathi, Todd Mytkowicz, Jacob Nelson, Olli Saarikivi, and Rachee Singh. 2023. TACCL: Guiding Collective Algorithm Synthesis using Communication Sketches. In *NSDI*.
- [45] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarsz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *arXiv:1701.06538 [cs.LG]*.
- [46] Mohammad Shoeybi, Mostafa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. In *arXiv:1909.08053 [cs.CL]*.
- [47] Alexander Shypula, Aman Madaan, Yimeng Zeng, Uri Alon, Jacob R. Gardner, Yiming Yang, Milad Hashemi, Graham Neubig, Parthasarathy Ranganathan, Osbert Bastani, and Amir Yazdanbakhsh. 2024. Learning Performance-Improving Code Edits. In *ICLR*.
- [48] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiuyan Hu, Harry Yang, Oron Ashual, Oran Gafni, Devi Parikh, Sonal Gupta, and Yaniv Taigman. 2022. Make-A-Video: Text-to-Video Generation without Text-Video Data. In *arXiv:2209.14792 [cs.CV]*.

- [49] Siddharth Singh, Olatunji Ruwase, Ammar Ahmad Awan, Samyam Rajbhandari, Yuxiong He, and Abhinav Bhatle. 2023. A Hybrid Tensor-Expert-Data Parallelism Approach to Optimize Mixture-of-Experts Training. In *ICS*.
- [50] Mellanox Technologies. 2003. Introduction to InfiniBand. https://network.nvidia.com/pdf/whitepapers/IB_Intro_WP_190.pdf.
- [51] Tesla. 2021. Tesla Dojo Technology. <https://cdn.motor1.com/pdf-files/535242876-tesla-dojotechnology.pdf>.
- [52] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (2005), 49–66. doi:10.1177/1094342005051521
- [53] Trieu H Trinh, Yuhuai Wu, Quoc V Le, He He, and Thang Luong. 2024. Solving Olympiad Geometry without Human Demonstrations. *Nature* (2024).
- [54] Amin Vahdat and Mark Lohmeyer. 2023. Enabling next-generation AI workloads: Announcing TPU v5p and AI Hypercomputer. <https://cloud.google.com/blog/products/ai-machine-learning/introducing-cloud-tpu-v5p-and-ai-hypercomputer>.
- [55] Leslie G. Valiant. 1990. A Bridging Model for Parallel Computation. *Commun. ACM* 33, 8 (1990), 103–111. doi:10.1145/79173.79181
- [56] Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, and Anthony Kewitsch. 2023. TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs. In *NSDI*.
- [57] William Won, Midhilesh Elavazhagan, Sudarshan Srinivasan, Ajaya Durg, Samvit Kaul, Swati Gupta, and Tushar Krishna. 2024. TACOS: Topology-Aware Collective Algorithm Synthesizer for Distributed Machine Learning. In *arXiv:2304.05301 [cs.DC]*.
- [58] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. 2023. ASTRA-sim2.0: Modeling Hierarchical Networks and Disaggregated Systems for Large-model Training at Scale. In *Proceedings of the 2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS '23)*. 283–294. doi:10.1109/ISPASS57527.2023.00035
- [59] William Won, Saeed Rashidi, Sudarshan Srinivasan, and Tushar Krishna. 2024. LIBRA: Enabling Workload-aware Multi-dimensional Network Topology Optimization for Distributed Training of Large AI Models. In *ISPASS*.
- [60] Liangyu Zhao, Siddharth Pal, Tapan Chugh, Weiyang Wang, Jason Fantl, Prithwish Basu, Joud Khoury, and Arvind Krishnamurthy. 2024. Efficient Direct-Connect Topologies for Collective Communications. In *arXiv:2202.03356 [cs.NI]*.