AIPerf: Meta's Performance Modeling and HW/SW Co-design Framework for Datacenter-Scale LLM Training

Pyeongsu Park*, Hwijoon Lim*, Seah Kim*†, Mustafa Ozdal*, Yi-Chien Lin*, Kevin Siu*, Pavan Yalamanchili*, Benjamin Ghaemmaghami*, Yufeng Gu*, Jongsoo Park*, Joel Coburn*, Paul Baumstarck*, Shobhit Kanaujia*, Chunqiang Tang*
*Meta Platforms †University of Michigan

***Abstract*—Co-designing large language models and infrastructure is a critical challenge at hyperscale. Because models evolve daily while hardware decisions require years of lead time, misalignments can cause significant inefficiencies in our AI ecosystem. This paper presents AIPerf, a performance-modeling framework designed to guide decisions throughout our six-phase model-hardware co-design lifecycle. We outline practical challenges at each phase—such as cross-platform competitive analysis, parallelism configuration, and workload-dependent rack decisions—and derive five core requirements for a production-grade modeling framework. We demonstrate how AIPerf meets these needs through five case studies, ranging from using headroom analysis to identify software optimization roadmaps to conducting what-if analysis for network topology and scale-up domain size. AIPerf has supported this lifecycle for years, directly informing our in-house AI chip design and frontier model training decisions.**

***Index Terms*—Generative AI, performance modeling, parallelism, mixture-of-experts, hardware-software co-design**

I. Introduction

Co-designing large language models (LLMs) and infrastructure is a critical challenge at hyperscale. Models evolve daily, yet hardware decisions, from chip to rack architectures to fleet procurement, must be made years in advance given the long design-to-deployment cycle. Meta's dual role as both a silicon designer and a hyperscale consumer provides us with unique opportunities. We co-design custom MTIA accelerators [1], [2] alongside platforms from external vendors, making planning decisions across a six-phase, multi-year lifecycle (§II-B) that spans heterogeneous hardware and software landscapes.

Making planning decisions trustworthy across this lifecycle is hard because the features that govern training performance are deeply intertwined. Memory feasibility, 5D parallelism, pipeline schedule, collective placement, topology hierarchy, and workload non-uniformity all interact, and modeling any single feature in isolation can be misleading. A chip architecture chosen during design constrains the parallelism strategies available during deployment; a topology provisioned for dense models may bottleneck mixture-of-experts (MoE) workloads; adding expert parallelism (EP) without modeling routing imbalance can lead to wrong decisions. A framework that does not model these interactions with high fidelity does not merely produce inaccurate numbers; it selects the wrong platform, configuration, or architecture, with consequences that compound across the fleet and across years.

The scale of these decisions, the heterogeneity of evaluation platforms, and the uncertainty of co-designing silicon years ahead of AI workloads make ad-hoc approaches insufficient. When a single wrong decision can misallocate datacenter-scale resources and months of engineering effort, every phase of the lifecycle demands a planner that is systematic, quantitative, and consistent across platforms. Meta utilizes such a framework at every phase. During architecture evaluation, our framework drives *competitive analysis* across internal MTIA designs and external vendor platforms, directly informing important procurement and manufacture commitments. During silicon bring-up, it provides performance targets that distinguish fundamental hardware limitations from temporary software inefficiencies, catching problems before the next tapeout rather than after mass production. During model co-design, rigorous *what-if analysis* lets teams explore performance trade-offs across architecture variants before committing to expensive training runs.

In this paper, we describe how Meta's hardware-AI co-design lifecycle relies on a production-level performance modeling framework to produce business-critical decisions at each phase, what requirements this lifecycle imposes, and how our framework, AIPerf, addresses them. AIPerf has served this lifecycle for multiple years, informing decisions from MTIA chip and rack architectures carried into fabrication to training configurations for frontier models. We present five case studies illustrating how jointly modeling compute, memory, and communication across the full parallelism space surfaces insights that would otherwise be missed: optimization headroom on deployed platforms, parallelism configurations that outperform expert-curated heuristics, why capturing MoE routing imbalance is important, workload-dependent topology trade-offs, and scale-up domain size decisions for MTIA rack architecture. Prior performance tools for distributed training [3]–[8] address individual aspects of this problem but treat

†This research was conducted during Seah Kim's visiting appointment at Meta. She led the publication effort and made pivotal contributions to the technical analysis and overall narrative of the manuscript.

Axis	Range or options	Impact				
		Compute efficiency	**Communication overhead**	**Pipeline bubble**	**HBM usage**	**Load imbalance**
Parallelism						
Tensor parallelism (TP)	1–16	✓	✓	–	✓	–
Pipeline parallelism (PP)	1–32	–	✓	✓	✓	–
Context parallelism (CP)	1–8	✓	✓	–	✓	✓
Expert parallelism (EP)	1–256	✓	✓	–	✓	✓
Parallelism ordering	permutations	–	✓	–	–	–
Schedule and memory knobs						
Microbatch size	1–16	✓	✓	✓	✓	✓
Pipeline schedule	BFS / DFS / ZeroBubble / DualPipe	–	✓	✓	✓	–
Sharding level	ZeRO-1 / 2 / 3	–	✓	–	✓	–
Recomputation	none / selective / full	✓	–	–	✓	–
Offloading	activations / weights / gradients / optimizer	–	✓	–	✓	–

TABLE I: Simplified training plan search space; precision knobs are omitted. ✓ indicates the axis affects the metric; – indicates no direct effect. Jointly, these axes yield $O(10^4)$ candidate plans per model–platform pair, and they interact—so the space cannot be searched one axis at a time.

them in isolation and target a single platform at a single point in time.

Specifically, this work makes the following contributions:

1) Drawing on Meta's experience as both a frontier-model trainer and a custom-silicon designer, we characterize the *AI infrastructure and model co-design lifecycle* and derive performance-modeling requirements, each grounded in production planning challenges (§II).
2) We present AIPerf's design addressing these requirements: joint 5D parallelism search with memory-feasibility pruning, schedule-aware timing, and a modular architecture for cross-platform retargeting. What makes AIPerf reliable for datacenter-scale hardware decisions is its broad model coverage (e.g., MoE with load imbalance) and its accounting for system dynamics (e.g., memory feasibility). Along the way, we share non-intuitive insights drawn from our production experience with AIPerf (§III).
3) We share *multi-year production deployment experience* across five case studies, each surfacing a planning insight that requires jointly modeling compute, memory, and communication across the full parallelism space (§IV).

II. Performance Modeling and Co-design at Meta

This section describes the properties that make hardware–AI co-design planning at Meta uniquely challenging (§II-A), Meta's co-design lifecycle and the role of performance modeling within it (§II-B), and derives five requirements that a production performance modeling framework (referred to as the "planner" in the rest of the paper) must satisfy, motivating the necessity of AIPerf (§II-C).

A. Hardware–AI Co-design Challenges

Co-designing LLMs and infrastructure for large-scale training is among the most complex planning challenges Meta faces today, driven by four properties.

First, the *scale and financial stakes* of LLM training are unprecedented. Training is throughput-driven: tens of thousands of accelerators train a single model over months, making efficiency a fleet-wide multiplier. A wrong decision in chip architecture, network topology, platform selection, or model design can result in datacenter-scale resource misallocation.

Second, Meta actively evaluates *multiple heterogeneous platforms* to identify the best total cost of ownership (TCO) under power and time constraints. Each platform differs in compute architecture, memory hierarchy, interconnect topology, and software stack, requiring that every candidate be assessed on equal footing under realistic production workloads rather than peak specifications.

Third, Meta's position as *both a silicon designer and a hyperscale consumer* amplifies both opportunity and risk: custom MTIA accelerators can be tightly optimized to our workloads, but a wrong bet is fully ours to bear, and multi-year fabrication commitments must anticipate model architectures that evolve daily.

Fourth, the hardware and performance trade-offs in LLM training are *tightly coupled across dimensions*. Parallelism, pipeline schedule, memory optimization, and precision interact, so using simplistic analysis can give misleading results. Worse, the space of choices is *combinatorially large*: the parallelism dimensions, together with schedule, memory, and precision knobs, yield $O(10^4)$ candidates per model–platform pair (Table I; detailed in §III-E).

Together, scale, platform heterogeneity, co-design uncertainty, and coupling make ad-hoc approaches insufficient and *demand a systematic framework*—what we call a *planner*.

B. Co-design Lifecycle

Performance planning for GenAI training is a recurring process tied to our hardware-AI co-design lifecycle. Each chip and platform generation follows a multi-year, six-phase pipeline (Figure 1): architecture design (P1), competitive analysis (P2), silicon design (P3), deployment (P4), model co-design (P5),

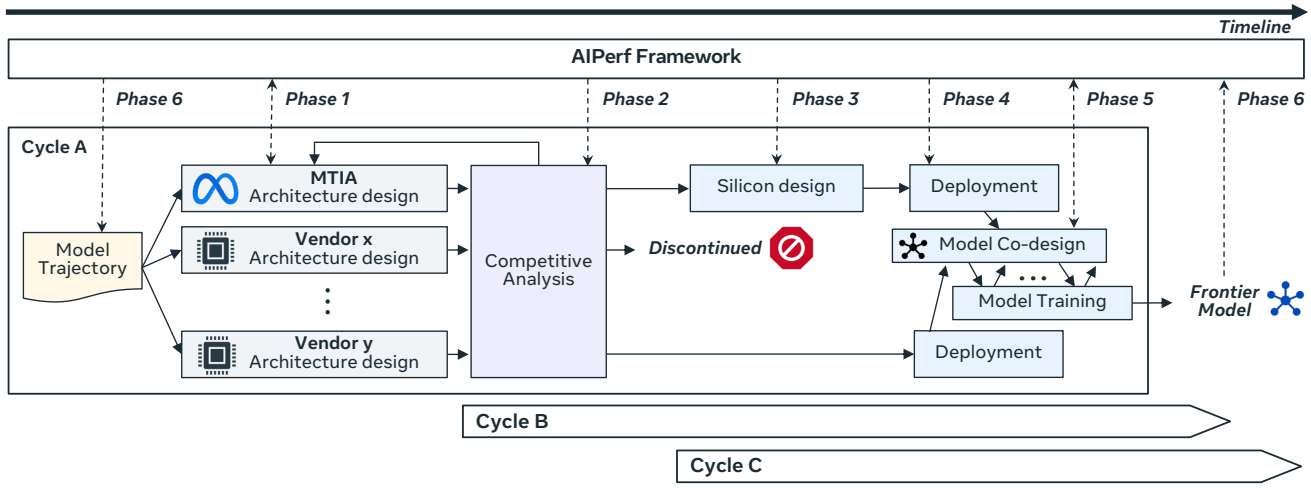


Fig. 1: Hardware-AI co-design lifecycle at Meta. Multiple chip and platform generations progress through six phases concurrently. Dashed arrows indicate feedback loops; platforms that fail competitive analysis are discontinued.

and trajectory setup (P6). Every phase produces decisions that affect efficiency at datacenter scale, and each poses unique practical challenges that the literature largely overlooks.

Three characteristics of this lifecycle shape what the planner must do. (i) All six phases run *simultaneously* across different platforms and model generations: one MTIA generation may be in architecture design while another is in silicon design and a GPU platform is in production training, each requiring concurrent support. (ii) The dominant challenge *shifts* across phases, from *uncertainty* in early phases (projecting performance for hardware and models that do not yet exist) to *time-sensitive, model-centric* decisions in later phases that demand rapid search over large configuration spaces. (iii) The lifecycle is not linear: *feedback loops* connect later phases back to earlier ones (e.g., a silicon design gap can revise the architecture before the next tapeout, and production measurements can reshape model co-design), so the planner must give consistent, comparable projections across phases for measured insights to trace back to planning assumptions.

Planner role overview. The planner serves three main roles: (1) *what-if analysis*, projecting how a new idea (from academia, industry, or internal research) affects training throughput and memory feasibility so teams can weigh trade-offs before committing to expensive training runs or hardware choices; (2) *prioritization*, ranking competing ideas by projected end-to-end impact so the highest-value changes are integrated first; and (3) *go/no-go recommendations*, flagging early when a candidate platform is uncompetitive under the target workload mix, where redirecting the program is far cheaper than discovering the shortfall after fabrication or deployment.

1) Phase 1: Architecture Design:
In this phase, Meta defines the high-level hardware specifications that will serve production models after deployment, working concurrently with multiple external vendors and internal MTIA teams to ensure alignment with projected workload requirements.

The phase involves two design activities. *Accelerator co-design* determines chip-level parameters: compute throughput, high-bandwidth memory (HBM) capacity, on-chip interconnect bandwidth, and memory architecture. *Infrastructure co-design* determines the system around accelerators: scale-up domain size, network topology, scale-out network, and power envelopes. These activities cannot be conducted independently: a change in HBM capacity can shift the optimal scale-up domain size, which affects interconnect provisioning.

The planner guides these decisions by projecting how accelerator-level characteristics translate to fleet-scale training throughput. For instance, when a vendor proposes a specific interconnect bandwidth or domain size, the planner quantifies the throughput consequence for our actual workload mix—not as a single number, but as a breakdown of exposed communication by parallelism type, pipeline bubble fraction, and per-operator compute efficiency. This breakdown transforms vendor discussions from qualitative trade-offs into quantitative arguments grounded in actual workload costs. We present AIPerf's architecture design trade-off analysis for the MTIA 400 platform [2] as case studies in §IV-D and §IV-E.

This planning is challenging because of the *uncertainty* inherent at this stage. Hardware simulators can model individual accelerator performance but are typically limited to small scale; they cannot project how chip-level choices affect end-to-end training throughput across tens of thousands of accelerators. Moreover, performance estimates at this phase often rely on analytical models that tend to be optimistic, as they do not capture practical inefficiencies such as software overhead, collective algorithm limitations, and workload non-uniformity. The planner must account for these non-idealities so that architectural decisions rest on realistic projections rather than best-case estimates.

2) Phase 2: Competitive Analysis:
Competitive analysis is the most consequential phase in Meta's

hardware program. While Phase 1 explores the design space for individual platforms, this phase compares finalized candidates apples-to-apples to determine which platforms advance to silicon design or procurement. We evaluate each candidate platform, including our own MTIA accelerators, across various dimensions such as end-to-end training throughput, total cost of ownership, power and thermal budget, and delivery timeline. These evaluations feed business reviews that determine datacenter-scale hardware procurement and the commitments we secure from manufacturers.

This level of systematic analysis is uncommon in the industry. It is essential at Meta because we simultaneously manage multiple vendor relationships and design our own accelerators: every platform must be evaluated under the same methodology with no exceptions—MTIA receives no preferential treatment—and negotiations must be grounded in workload economics rather than peak specification comparisons. Our planner has informed competitive analysis decisions across many hardware platforms, such as MTIA and NVIDIA GPUs. It is used both to compare across vendors and to weigh closely related offerings within a single vendor's lineup (e.g., B200 vs. GB200 vs. GB300), identifying the best hardware configuration for Meta's workloads.

The central challenge is ensuring *fairness* across heterogeneous platforms. An apples-to-apples comparison is nontrivial because each platform differs in compute architecture, memory hierarchy, interconnect topology, and software stack. The planner must apply consistent workload assumptions while allowing each platform to be evaluated under its own best parallelism configuration, so that performance differences reflect genuine hardware characteristics rather than planner artifacts.

3) Phase 3: Silicon Design:
This phase is unique to MTIA, as Meta fabricates its own accelerators. Once a platform passes competitive analysis, the architecture is refined into real silicon through a series of design milestones that culminate in tapeout, each representing an increasingly committed stage of chip design.

At each design milestone, the planner provides performance targets against which the design is validated. As cycle-level and register transfer level (RTL) simulation results become available, they replace the analytical estimates used in earlier phases. The planner translates these chip-level results into end-to-end training throughput at fleet scale, answering questions such as: does the chip meet the projected compute throughput, memory bandwidth, and interconnect latency under representative training workloads? Early power analysis also becomes available at this stage, enabling the planner to validate thermal constraints and refine rack-level design decisions.

Critically, if the planner identifies a severe gap (a microarchitecture-level issue whose end-to-end impact cannot be mitigated through software optimization), the *feedback loop returns to Phase 1*. The architecture can be revised before the next tapeout, avoiding the cost of fabricating and deploying a suboptimal design. A gap that is instead resolvable via software or kernel optimization becomes a concrete, quantifiable target. Catching a design issue at this stage saves months of schedule and tens of millions of dollars compared to discovering it after mass production; this early detection capability is one of the highest-value uses of the planner.

Our planner has been used to rank detailed microarchitectural options for the MTIA platforms, such as memory hierarchy bandwidth and capacity trade-offs, as well as related power decisions such as clock frequency and thermal design power (TDP).

4) Phase 4: Deployment:
After a chip is taped out, the platform enters production and fleet deployment. Deployment is a continuous process: the system is not fully operational immediately since capacity is scaled and optimizations are implemented over time. As a result, the subsequent phases of model co-design and training begin in parallel, ramping alongside the growing fleet.

The planner serves two purposes during deployment. First, it provides *validation targets* for product introduction while the framework and runtime support for new silicon becomes available but is not yet fully optimized. When validation reveals performance gaps, such as a collective running slower than modeled or a compute kernel falling below its target, the planner quantifies the end-to-end impact to guide optimization efforts. Second, the planner establishes *long-term optimization targets* aligned to launch timelines, so that engineering effort is prioritized by projected throughput impact.

The central challenge is *diagnosing the root cause* of performance gaps. When measured step time exceeds the planner's projection, the gap must be attributed to a specific source: a fundamental hardware limitation, an unoptimized kernel, an inefficient collective algorithm, or framework-level overhead.

This is where our planner departs from typical performance models. In a production setting where the software stack is updated weekly, any point-in-time calibration goes stale rapidly, making absolute accuracy a moving target.

We find that two other capabilities matter more. One is *plan-ranking fidelity*. A planner that consistently ranks configurations correctly is more useful than one that matches wall-clock time but mis-orders candidates. The other is *gap attribution*: when measured performance falls short, the planner must pinpoint which component is responsible so that optimization effort is directed effectively. We deliberately model the ideal case so that discrepancies between prediction and measurement reveal software optimization headroom rather than environmental noise.

This process is inherently iterative. As fleet capacity increases, new scale-dependent bottlenecks emerge, and feedback from later phases (model co-design and training) introduces additional optimization targets. The planner is re-invoked at each iteration, updating projections and re-prioritizing efforts as the platform matures.

At Meta, the planner has been used to set critical optimization targets when a new platform is introduced: it identifies, for our internal and external partners, the optimizations expected to deliver the largest gains for our workload mix, along with

the projected gain for each, so that partners can prioritize and iterate quickly with quantitative feedback.

5) Phase 5: Model Co-design and Training:
After a platform is deployed, the lifecycle enters its most iterative phase: co-designing the model architecture with the platform and selecting production training configurations. Machine learning researchers and engineers concurrently explore multiple design axes, including attention mechanisms, parallelism strategies, kernel implementations, numerical precision, and model architecture choices, seeking configurations that maximize training efficiency on the target platform. This exploration, combined with ongoing production training, often spans months per model generation.

The planner enables this exploration through *what-if analysis*, helping model designers and system engineers reason about performance trade-offs before committing to expensive training runs. Typical questions include: what is the impact of increasing expert count on memory and communication? Can the model train at a longer context length on the target cluster? If the expert feed-forward network (FFN) dimension is increased, does the configuration still fit in memory?

The challenge is that the design axes are both *deeply intertwined* and *rapidly evolving*. They interact: adding expert parallelism without modeling routing imbalance gives wrong plan rankings, and adding pipeline parallelism without schedule-aware timing misses bubble costs that can dominate step time. And they move fast—model architectures change daily, so the planner must give rapid feedback across variants without being tightly coupled to any specific model version. At Meta, we have used the planner to evaluate the performance impact of software optimizations and model-architecture choices for our frontier models, for example, parallelism strategies, memory-management algorithms, and hyperparameter sweeps.

6) Phase 6: Trajectory Setup:
Unlike the preceding phases, trajectory setup is not a discrete operational step but a forward-looking planning activity that shapes future hardware-AI co-design cycles. We described the earlier phases against a fixed target model for clarity, but in practice the model is neither stable nor known in advance; the hardware being designed will ultimately serve models that do not yet exist. Trajectory setup addresses exactly this. Overfitting hardware decisions to the current model generation is a recurring risk: for example, an MTIA design optimized for Llama 3, a dense model, may be poorly suited to Llama 4, which adopts mixture-of-experts (MoE) with different compute, memory, and communication characteristics.

To mitigate this risk, we establish *model trajectories*, which are projected workload mixes that represent where model architectures are heading over a multi-year horizon. These trajectories are informed by scaling trends of current models, emerging research directions, and feedback from stakeholder teams. The planner evaluates hardware design decisions against these trajectories rather than any single model, using aggregate metrics across the projected workload mix to ensure recommendations are robust to model evolution. For each candidate hardware configuration, this aggregate is an importance-weighted combination of per-workload results, each computed under the workload's own best execution setup, so designers can pick the single best compromise configuration (§IV-E).

This phase is the core of the *cross-cycle feedback loop*: insights from production training and model co-design feed back into trajectory assumptions, which in turn reshape architecture design decisions for the next hardware generation.

C. Derived Requirements

The six phases above establish *when* and *why* a performance modeling framework is needed; we now turn to *what* it must get right. Training performance is governed by intertwined features: parallelism configuration, pipeline schedule, collective placement, memory optimization, and workload non-uniformity. Modeling any single feature in isolation produces projections that are not merely approximate but actively misleading.

From this observation and our multi-year operational experience, we derive five requirements (R1–R5) that a production planner must satisfy simultaneously. Each requirement is grounded in the concrete challenges of one or more phases. Together, they form the requirement set against which we evaluate AIPerf and existing tools. These are *modeling* requirements, not the complete business objective: downstream decision processes apply cost, power, schedule, reliability, and procurement constraints on top of AIPerf's performance projections (§IV-D, §IV-E).

R1: Joint multi-dimensional plan search and validation (P1–P6)**.** The planner must jointly enumerate the full parallelism and configuration space, validate each candidate against memory capacity, and rank survivors by projected throughput. The five parallelism dimensions (DP, TP, PP, CP, EP) interact non-linearly, so axis-by-axis tuning is unreliable and even misleading: a plan that appears optimal along any single axis may be far from optimal when all dimensions are considered jointly (we detail the EP/TP/memory coupling in §III-E). Sharding level, microbatch size, recomputation granularity, and offloading targets [9]–[12] multiply the space further, and memory-infeasible configurations must be re-evaluated with each recovery knob's performance cost included. This joint search recurs across the lifecycle: every platform and chip revision must be evaluated under its best configuration, and deployment and model co-design require rapid re-search whenever a change shifts the optimum.

R2: Hardware-agnostic model representation and scheduling (P1, P2, P4, P5)**.** For each candidate configuration that R1 generates, the planner must project its performance without embedding hardware-specific assumptions, so that performance differences reflect hardware, not planner artifacts. This requires a hardware-independent intermediate representation that captures model computation and communication semantically, and a parallelism-aware scheduler that includes microbatch-level pipeline timing across multiple schedule families (e.g., BFS, DFS, ZeroBubble [13], DualPipe [14]), since two configurations with identical FLOPs and communication volume can have materially different step times depending

on the schedule [15]–[17]. Without this portability, comparing platforms becomes comparing planner implementations rather than hardware.

R3: Modular and extensible architecture (P1–P5)**.** The planner must absorb new hardware platforms, network topologies, pipeline schedule families, model architectures (including novel attention and routing mechanisms), and memory optimization techniques without rewriting core planning logic. A planner that requires re-engineering for each new element cannot keep pace with a lifecycle where such changes arrive continuously across platforms and models.

R4: Production workload calibration (P1–P6)**.** Production workloads diverge from idealized assumptions in two ways the planner must account for. First, *workload non-uniformity*: in MoE architectures, token routing produces skewed expert loads, causing straggler ranks to dominate step time [18]. A planner that assumes uniform workloads will select configurations that appear optimal in theory but perform poorly under real routing distributions. This risk affects every phase from architecture provisioning (P1) through production plan selection (P4), and it shapes the workload mixes used for trajectory evaluation (P6). Second, on deployed platforms, runtime factors such as dynamic memory overhead, power behavior, and contention fall outside steady-state modeling; calibrating for these is essential during deployment (P4), where the gap between projection and measurement reveals optimization headroom.

R5: Cross-platform hardware portability (P1–P5)**.** The planner must retarget to new hardware platforms without modifying the workload representation or planning logic. These platforms span pre-silicon designs evaluated via analytical models or cycle-accurate simulators, partially enabled accelerators with only kernel-level microbenchmarks, and fully deployed systems with production benchmarks. Retargeting must also account for platform maturity: directly comparing simulator-derived estimates against production benchmarks is misleading, and even across deployed platforms, the degree of software optimization varies by generation. The planner must normalize for these maturity differences so that cross-platform comparisons reflect intrinsic hardware characteristics rather than software stack maturity (together with R2, this ensures apples-to-apples comparison).

Taken together, R1–R5 define a combined requirement set that no prior tool was designed to satisfy jointly.

D. Why a Custom Framework?

No existing tool was designed to jointly satisfy R1–R5. Each prior approach addresses only a subset, and in our experience extending one incrementally falls short: piecemeal additions miss the cross-feature interactions (R1, R4) and lack the portable, extensible architecture needed for continuous cross-platform retargeting and rapid software stack evolution (R3, R5). This motivated AIPerf as a purpose-built framework, described next. §III-G and §V provide in-depth comparisons of Meta's planner and representative tools across all requirements.

III. AIPerf: Meta's Co-design Planner

AIPerf has served all six phases in production for multiple years, informing decisions from training configurations for frontier models to chip and rack architecture for Meta's MTIA accelerator program [1], [2], where its analyses shaped compute/memory specifications, domain sizing, and interconnect choices carried into fabrication. Inference and recommendation-model planning are supported but out of scope for this paper. For Meta's approach to inference configuration design, we refer readers to our concurrent work [19].

A. Motivating Examples

AIPerf routinely produces conclusions that contradict initial intuition. For example, let us consider the decision of choosing the scale-up domain size (the number of accelerators linked by a high-bandwidth interconnect, e.g., over NVLink) for a future hardware platform. Our initial intuition was that larger scale-up domain sizes would enable efficient training of future MoE models with a larger number of experts. However, AIPerf analysis showed that several dimensions must be weighed jointly:

- **HBM bandwidth and capacity**: We can keep the degree of expert parallelism (EP) small even if the number of experts increases. However, letting each GPU be responsible for more experts would not only require more HBM capacity but also more HBM bandwidth due to the reduced arithmetic intensity of FFN operations.
- **Inter-datacenter (DC) network**: Large-scale training requires data parallelism across DCs, which means all gradients need to be periodically synchronized across all DCs. Increasing the number of experts can lead to inter-DC communication becoming a bottleneck because more weights need to be communicated. Inter-DC bandwidth can thus limit how much we can increase the number of experts before we get any benefit from a larger scale-up domain.
- **Software optimizations**: Top-k routing has a factor of k redundancy in the EP communication. This allows a two-step EP communication, where only a single copy of each token is sent through the scale-out network, and k copies are distributed within scale-up domains [14]. In addition, EP communication can be overlapped with FFN computation through software pipelining. Such optimizations can reduce the potential benefits of a larger scale-up domain size.
- **Data dependency**: Token imbalance across experts can lead to load imbalance across GPUs. Assigning more experts per GPU (i.e., reducing the degree of EP) can reduce this imbalance due to an averaging effect. A larger scale-up domain allows larger degrees of EP, but the benefits of a larger EP can be offset due to the increased load imbalance.

B. AIPerf Overview

AIPerf is designed around one principle: the same planning logic must produce trustworthy, comparable projections across platforms at any stage of maturity, from conceptual future hardware to fully deployed production systems. AIPerf (Figure 2) decouples *performance models* (§III-C) from a

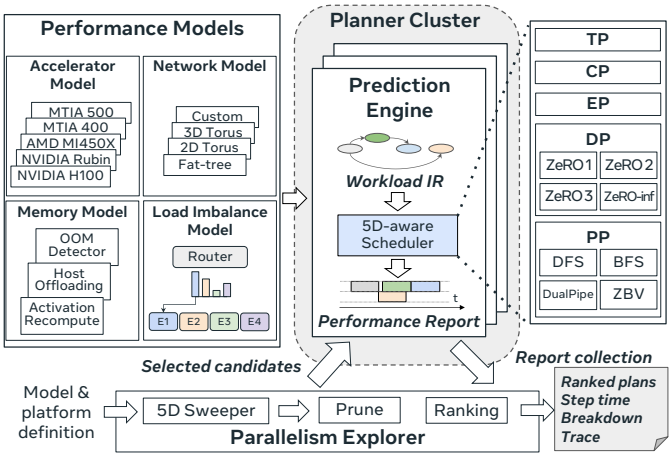


Fig. 2: AIPerf architecture. The parallelism explorer sweeps the 5D configuration space, prunes infeasible candidates via the memory model, and reports projections derived from platform-specific performance models.

graph-based *performance prediction engine* (§III-D). The performance models capture platform- and model-specific characteristics (accelerator throughput, network topology, memory capacity, and workload non-uniformity) while the prediction engine operates on a hardware-agnostic workload graph, scheduling components across parallelism dimensions and projecting end-to-end step time. This separation is what makes fair cross-platform comparison possible (R2, R5): platform-specific details are confined to the models, while the scheduling and projection logic remains identical across platforms. The *Parallelism Explorer* (§III-E) drives this pipeline (R1), sweeping the combinatorial configuration space of Table I by requesting projections for each candidate plan, pruning infeasible configurations, and ranking survivors. Production calibrations (§III-F) account for runtime dynamics (memory guardbands and runtime performance refinement) that fall outside steady-state modeling (R4). Exploration is CPU-intensive at this scale, and it must run continuously as models and platforms co-evolve, since decisions are time-sensitive. Meta dedicates a cluster of more than 10,000 CPU cores to AIPerf, ranking millions of configurations across platforms and model variants. The following subsections detail each component and show how the requirements from §II are addressed.

C. Performance Models

AIPerf's performance models capture the platform- and workload-specific characteristics that the prediction engine cannot derive from the workload graph alone: how fast each operator executes on a given accelerator, how collectives behave across the network topology, whether a configuration fits in memory, and how workload non-uniformity affects per-rank execution time. Each model is independently maintainable and pluggable (R3), so that a new platform or a new source of non-uniformity can be incorporated without modifying the prediction engine or the exploration pipeline. The following subsubsections describe the four models.

1) Accelerator-Performance Model: The accelerator-performance model is AIPerf's primary mechanism for incorporating platform-specific performance characteristics (R5). An accelerator specification alone does not predict how operations perform at fleet scale, and simple analytical models are typically too optimistic for business-critical decisions. The model is therefore *hybrid*: it pairs analytical reasoning with an empirical performance database, supplemented by recipes that encode expert knowledge and cross-platform calibration. For each query from the prediction engine, the model takes a Component's operation type, tensor shape, and precision and returns a predicted execution time. Crucially, the model does not represent each platform's microarchitecture directly; microarchitecture-level behavior is captured empirically in the database and adjusted by the recipe layer, so the prediction engine itself remains microarchitecture-agnostic.

We decouple the prediction engine from microarchitectural detail deliberately. We evaluate many platforms from internal and external vendors concurrently, so maintaining a detailed microarchitectural model per platform is infeasible and would substantially inflate projection time, which is already significant given the size of the configuration space (Table I). Microarchitecture-level simulators are also typically proprietary: vendors expose performance rather than microarchitectural details, sometimes without disclosing whether the numbers are analytical, cycle-accurate, or measured, and they tend to model GEMM kernels well while being far less accurate for other kernels and for system-level inefficiencies. Directly comparing such simulator-derived estimates against real benchmarks is therefore misleading, especially when contrasting current and future platforms.

We realize this hybrid design through three elements: a performance database that supplies measured data, a recipe layer that maps and calibrates it per platform, and benchmark-directed power modeling.

Performance database. Because the Parallelism Explorer sweeps a wide range of configurations, the underlying kernel shapes vary widely, so we maintain an extensive performance database. It draws from analytical models, cycle-accurate simulators, vendor specifications, and execution traces, combining them pragmatically based on what each platform provides at its maturity stage. Benchmarking spans kernel-level, component-level, and end-to-end (often trace-based) granularities, whose availability varies substantially from project to project. Non-GEMM operations often take non-negligible time, so the database deliberately captures a broad range of operations rather than GEMMs alone. It is refreshed periodically to track software-stack updates, hardware revisions, and evolving model architectures.

Recipe layer. An extensive database alone does not guarantee fair cross-platform comparison: the degree of software optimization varies by platform maturity and generation, and projecting from the current software stack risks undervaluing a platform whose addressable inefficiencies have not yet been resolved. We therefore introduce a *recipe* layer on top of the

raw database (R5). A recipe encodes how a Component maps to low-level operations on a specific platform and applies calibration that normalizes for maturity differences. For a pre-silicon platform, for example, recipes adjust simulator-projected (or roofline) performance using scaling factors derived from the trajectory of that vendor's previous hardware generations and our deployment experience with their earlier platforms; they also incorporate vendor commitments (e.g., a guarantee to reach a stated fraction of proposed performance at a given confidence level) and retrospective calibration against measured numbers. Recipes can further extrapolate one platform's performance from another to capture system-level inefficiencies. Production traces are central here, since for some models individually trivial operations (e.g., memcpy) dominate in aggregate. Recipes are expert-curated, vetted through extensive validation, and become increasingly important as platforms mature. Authoring a recipe is the only step required to retarget AIPerf to a new platform (R3, R5); the workload graph and prediction engine remain untouched.

Benchmark-directed power modeling. Power is a first-class consideration when building the database, increasingly so as accelerators become more power-constrained. AIPerf incorporates power at the TDP level for planning, and the database is parameterized by power-related variables spanning TDP levels and operation duty cycles, curated by internal and external experts to reflect realistic operating scenarios. Recent hardware, however, exposes sub-iteration power management (e.g., power excursions) that can shift per-kernel throughput; for existing platforms we calibrate these effects by adjusting the duty cycle at the benchmarking level, and for future platforms we collaborate with vendors to model anticipated power behavior during the design phase. AIPerf supports power optimization with a designer in the loop rather than by autonomous search: designers propose candidate chip- and rack-level TDP settings, and AIPerf reports the achievable performance for each, letting them identify the best-performing setup.

2) Network-Performance Model: AIPerf models the physical network as a hierarchical composition of *domains* (R5), for example, a flat scale-up domain (e.g., NVLink within a node) bridged by a lower-bandwidth scale-out fabric across the cluster. Parallelism dimensions are mapped onto this hierarchy by priority: TP at the innermost level, then CP/EP, PP, and DP. The key modeling decision is whether a parallel group fits within a single scale-up domain or crosses a domain boundary: in the former case, collectives access high-bandwidth, low-latency links; in the latter, traffic is routed over the scale-out fabric at a corresponding bandwidth penalty. Group placement and the ordering of parallelism dimensions are themselves search-space axes explored by the Parallelism Explorer (§III-E).

When multiple collectives from different parallelism dimensions execute concurrently, the topology also determines whether they contend for the same physical links or can overlap independently.

The topology model is pluggable (R3): AIPerf ships with fat-tree, 2D torus, and 3D torus backends, and adding a new topology requires only specifying the domain hierarchy and per-link bandwidths. This extensibility is what enables hardware co-design: we can project the impact of alternative interconnect structures and domain sizes on the same workload and observe how the optimal parallelism strategy shifts (§IV-D).

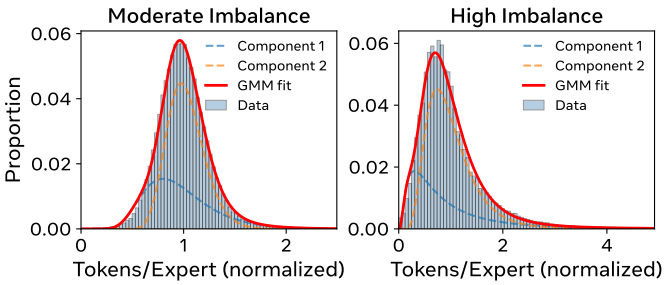


Fig. 3: Per-expert token count distributions from two MoE layers of Qwen-3 235B. AIPerf closely models the distribution through a two-component log-normal Gaussian Mixture Model.

3) Memory Model: AIPerf's memory model tracks peak HBM usage per pipeline-parallel rank, accounting for parameters, gradients, optimizer states, activations, and ZeRO-style sharding across data-parallel (DP) ranks [11], [20]. It also models the memory savings and performance cost of activation recomputation [9], [10] at configurable granularity and offloading to host RAM via PCIe or C2C [12] (R1).

To estimate activation memory, AIPerf extracts activation flow from the component graph and automatically determines whether each intermediate tensor should be stored or recomputed based on backward-pass necessity, inserting recompute nodes where needed. This enables accurate activation memory estimation for new architectures without manual per-operator annotation. The Parallelism Explorer (§III-E) uses this model to prune memory-infeasible configurations (R1).

4) Load-Imbalance Model: Production training workloads exhibit various sources of non-uniformity across ranks that can create stragglers dominating step time (R4). AIPerf provides a modeling framework for capturing these non-uniformities; here we focus on MoE routing imbalance, the most impactful case in our production experience (§IV-C).

In MoE model architectures, token routing produces non-uniform distributions across experts [18], causing both computation time and AllToAll communication volume to vary across ranks. A planner that assumes uniform routing will underestimate the straggler's execution time and overestimate the effective AllToAll bandwidth, selecting plans that appear optimal on paper but perform poorly under real routing distributions.

AIPerf models MoE routing imbalance by fitting a two-component log-normal Gaussian Mixture Model (GMM) to per-layer router statistics (per-expert token assignments per micro-batch) collected from training runs. The two compo-

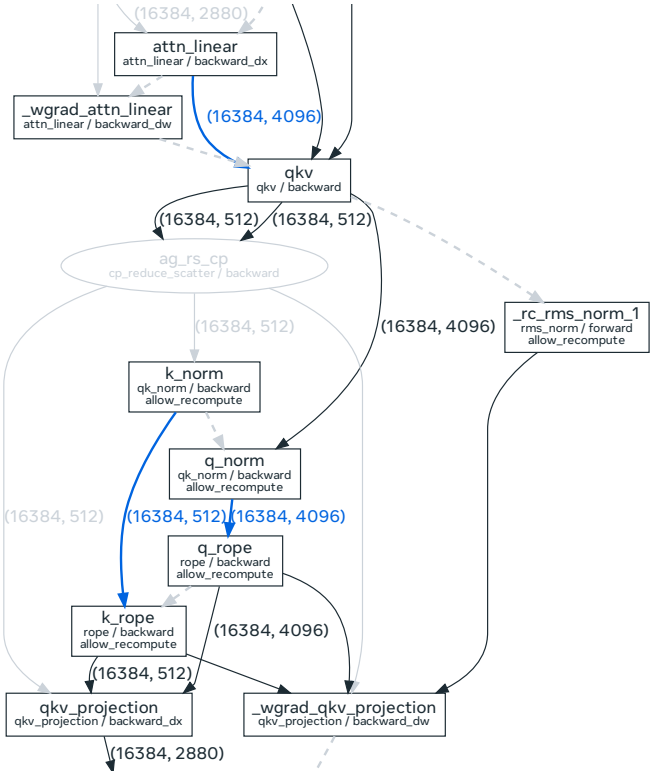


Fig. 4: Layer graph visualization (attention block excerpt). Each node is a Component; colored edges indicate stored activations, `_rc`-prefixed nodes denote recomputation.

nents capture both the majority of experts that receive similar token counts and the tail that receives disproportionately more. As Figure 3 shows, the GMM closely matches the empirical distribution across layers with varying degrees of skew. The fitted GMM parameters are compact and need to be profiled only once per model architecture, and can be reused across parallelism configurations and cluster sizes without re-profiling. At estimation time, AIPerf samples per-expert token counts from the per-layer GMMs to derive the straggler's extended MoE FFN compute time and asymmetric AllToAll volumes, with deduplication when multiple experts share the same device or node.

D. Performance Prediction Engine

Given a candidate plan and a target platform, the prediction engine derives the per-rank workload from the component graph, queries the performance models for per-component execution times, and evaluates the training iteration to project step time (R2). The engine supports 5D parallelism natively through a graph-based workload representation and internal schedulers, described below.

1) Platform-Agnostic Workload Representation: The workload representation captures *what* a model computes and communicates without prescribing *how* it executes on any particular hardware (R2). A model specification is translated into a hierarchical directed acyclic graph (DAG) of abstract *Components*, our hardware-agnostic intermediate representation (IR). Each Component describes a single logical operation (e.g., FFN, Input Projection, Attention, or collectives such as AllGather) in terms of its semantics: operation type, tensor shapes, and precision, with no kernel implementation, device-specific schedule, or software-stack assumption attached. Figure 4 shows an example of a Component graph.

Because the IR is defined at the semantic level rather than at the level of framework operators or hardware kernels, it provides two properties that the prediction engine relies on. First, components are *portable across hardware platforms* (R2, R5): the optimal kernels for a grouped GEMM or attention differ radically between an NVIDIA GB300 and a Meta MTIA chip and change whenever the software stack is updated [21], [22], yet the workload graph is untouched; only the performance model entry changes. Portability is why retargeting AIPerf across four MTIA tapeouts and multiple GPU generations required no change to the prediction engine (§IV-A). Second, components are *reusable across model architectures* (R3): an attention component is shared by every Llama variant and by other grouped-query-attention (GQA) architectures, differing only in shape parameters. This composability is how AIPerf scales to 90+ architectures, including all published and internal Llama models as well as open-source models such as GPT-OSS, DeepSeek-V3, and Qwen. New architectures are added by composing existing components or introducing a small number of new ones via graph templates, without modifying the core engine.

2) Parallelism-Aware Scheduler: AIPerf treats all five parallelism dimensions, Data (DP), Tensor (TP), Pipeline (PP), Context (CP), and Expert Parallelism (EP), as first-class axes rather than layering them via analytical approximations atop a single-device model (R2). Given a parallelism configuration, the engine derives the complete per-rank workload: TP, CP, and EP degrees determine per-rank tensor shapes, FLOPs, and memory footprint, while parallelism boundaries determine where synchronization collectives (AllReduce, AllGather, ReduceScatter, AllToAll) are needed and their data volumes. Each component's execution time is then obtained by querying the performance models with its operation type, tensor shape, and precision.

Among parallelism dimensions, PP stands out because each pipeline stage executes a different schedule of forward, backward, and communication operations, introducing bubbles, overlap patterns, and memory requirements that a simple analytical model cannot capture. This complexity is amplified in models that interleave structurally different layers, for example alternating full attention and sliding window attention, making per-stage computation non-uniform. AIPerf evaluates the pipeline schedule at the microbatch level (R2), supporting multiple pluggable schedule families including 1F1B (BFS and DFS) [16], [17], ZeroBubble [13], and DualPipe [14]. Adding a new PP algorithm is lightweight (R3): it requires only a specification of pipeline phases (e.g., warmup, steady-state, teardown) and microbatch ordering.

Model	HW	Configuration								Step	Per Layer Runtime (%)								Kernel Runtime (%)							
		TP	ETP	CP	PP	EP	Seq	mbs	gbs		Fwd SWA		Fwd FA		Bwd SWA		Bwd FA		Fwd SWA		Fwd FA		Bwd SWA		Bwd FA	
	(#GPU)									(A/M)	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A
Llama-3	GB300 (64)	2	2	1	1	1	8K	2	64	1.048	–	–	0.30	0.30	–	–	0.91	0.92	–	–	0.26	0.27	–	–	0.85	0.91
	H100 (64)	4	4	1	1	1	8K	2	32	1.014	–	–	0.34	0.34	–	–	0.84	0.88	–	–	0.26	0.26	–	–	0.74	0.84
Llama-4 Scout	GB300 (64)	2	1	1	2	4	16K	1	64	0.888	0.28	0.28	0.30	0.29	0.60	0.61	0.76	0.72	0.19	0.25	0.20	0.24	0.58	0.60	0.74	0.71
	H100 (256)	8	1	2	1	16	4K	1	16	0.941	0.23	0.23	0.23	0.22	0.42	0.39	0.42	0.39	0.09	0.09	0.09	0.09	0.26	0.27	0.25	0.27
GPT-OSS	GB300 (64)	2	1	1	1	8	16K	2	64	0.972	0.78	0.70	0.90	0.87	1.44	1.60	2.07	2.24	0.56	0.57	0.68	0.72	1.49	1.48	1.77	2.07
	H100 (256)	1	1	1	2	32	4K	1	512	0.924	0.36	0.37	0.36	0.50	0.63	0.68	0.66	0.56	0.17	0.18	0.18	0.19	0.59	0.54	0.60	0.58
Qwen-3	GB300 (64)	4	1	2	1	8	16K	2	16	0.898	–	–	0.27	0.28	–	–	0.77	0.77	–	–	0.19	0.22	–	–	0.62	0.68
	H100 (256)	4	1	2	1	32	4K	1	32	0.942	–	–	0.37	0.37	–	–	0.70	0.67	–	–	0.09	0.09	–	–	0.26	0.29

TABLE II: End-to-end validation across eight (model, platform, configuration) points: measured (M) vs. AIPerf (A) runtime (% of step time). ETP: expert tensor parallelism; SWA/FA: sliding-window/full attention layer; Seq: sequence length; mbs/gbs: micro-/global batch size; Step (A/M): AIPerf step time normalized to measured.

E. Parallelism Explorer

Finding the right training configuration is the central planning challenge that motivates AIPerf. As Table I shows, the search space—parallelism degrees together with sharding level, microbatch size, PP schedule family, recomputation granularity, and offloading targets—yields $O(10^4)$ candidates or more per model–platform pair. Crucially, these axes interact: changing EP degree shifts AllToAll volume, which changes the optimal TP degree, which affects memory pressure, which determines whether recomputation is needed, making axis-by-axis tuning unreliable. Without the Parallelism Explorer, teams must navigate this space through manual trial-and-error sweeps on live GPUs (§IV-B).

AIPerf replaces this with exhaustive enumeration over the full space (R1), made tractable by a three-stage cascading filter. First, algebraic constraints prune invalid mappings (e.g., TP degrees that do not evenly divide attention heads, EP degrees incompatible with expert count). Second, the memory model (§III-C3) eliminates configurations that exceed HBM capacity, while recovering plans that become feasible with recomputation or host-memory offloading and accounting for the performance cost of each knob (R1). Third, surviving candidates are evaluated by the prediction engine (§III-D), which projects step time and produces a bottleneck breakdown attributing iteration time to exposed communication, pipeline bubbles, and compute per operator, along with a per-component execution trace. Each evaluation is independent, so candidates are evaluated in parallel. Plans are ranked by throughput, and the accompanying breakdown tells teams not just *which* plan is best, but *why*.

F. Production Calibration

AIPerf deliberately models a single steady-state training iteration under ideal cluster conditions (R4). When measurement diverges from projection, we incorporate the behavior into the model only if it reflects a genuine hardware or model characteristic; if it is a software inefficiency, the model stays idealized. Discrepancies then reveal software optimization headroom and hardware bottlenecks, which is how our teams prioritize engineering effort (§IV-A). Two production factors still need explicit handling.

Memory guardband. Common libraries (e.g., communication stacks) allocate workspace buffers, and transient kernels reserve additional memory whose footprint is not statically known at planning time. AIPerf therefore caps usable HBM at 70–80% of physical capacity, reserving a guardband for these dynamic allocations. This cap is recalibrated periodically against measured runtime memory profiles to track evolving library behavior.

Runtime performance refinement. Steady-state projections do not perfectly capture runtime dynamics such as network contention, host-side orchestration overhead, checkpointing I/O stalls, and thermal throttling. We refine AIPerf's performance models after production through platform-specific performance characterization, while still taking into account potential software optimizations.

The calibration mechanisms above are simple yet sufficient for current platforms, but in our experience these effects intensify on platforms that push higher throughput under tighter power budgets, where lightweight calibration is increasingly stretched. We see several open problems here that we expect to grow in importance and where the broader community is well positioned to contribute.

One problem is characterizing the performance impact of power throttling and the stragglers it induces—a dynamic that becomes harder to model as accelerators grow more power-constrained. Another stems from the widening gap between per-generation GPU and CPU improvements, which makes host-side orchestration and synchronization overheads an increasing bottleneck. A third concerns the scalability of benchmarking and configuration search: as systems and models grow more heterogeneous, the search space becomes correspondingly broader and more complex. While our current methodology suffices for today's setups, we expect future platforms and models will demand more effective search-space pruning and richer modeling of precision-related behavior.

Encouragingly, many of these questions can be studied at modest scale, and we hope that frameworks like AIPerf and shared problem formulations help make them accessible beyond hyperscale settings.

G. Evaluation

We continuously validate and calibrate AIPerf against various workloads and platforms as part of its daily use in production hardware and model planning. Below we present

Scenario	AIPerf	vTrain
Modern model (MoE, long seq.)	Yes	No
Best config search	TP4/PP1/DP16	TP2/PP1/DP32 (OOM)
Accuracy, TP4/PP1/DP16*	0.94×	0.72×
Accuracy, TP8/PP1/DP8*	0.85×	0.61×

TABLE III: AIPerf vs. vTrain [7] on Llama-3 70B (H100). *Predicted step time normalized to measured.

Model	Params	# Experts	Top-k	d_model	d_hidden
Llama-3 70B [23]	70 B	Dense	–	8192	28672
Llama-4 Scout [24]	109 B	MoE (16)	1	5120	8192
GPT-OSS 120B [25]	120 B	MoE (128)	4	2880	2880
Qwen-3 235B [26]	235 B	MoE (128)	8	4096	1536

TABLE IV: Models used for case studies.

validation examples against open-source models and compare with a prior modeling tool.

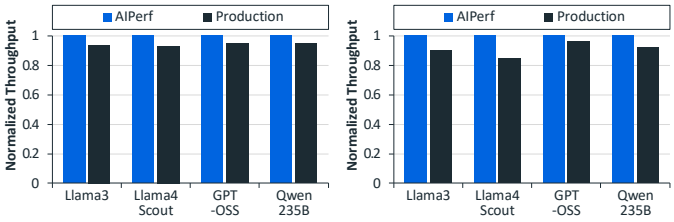
End-to-end accuracy. Using an internal trainer, we measure the performance of four models in Table IV on H100 and GB300 across eight parallelism configurations (Table II). We report the AIPerf-to-measured ratio (A/M) and per-component runtimes as a fraction of step time. AIPerf tracks measurements closely, averaging 6.2% absolute step-time error (worst case 11.2%, Llama-4 Scout GB300).

Comparison with prior modeling. To evaluate AIPerf's coverage and accuracy, we compare against a state-of-the-art, profiling-driven training simulator (vTrain [7]). For a fair comparison, we extended it with modern dense-LLM features (e.g., GQA, FlashAttention) and profiled a single decoder's kernels on H100.

Table III summarizes the comparison. First, AIPerf covers modern LLM architectures and longer sequence lengths through its MoE support and context-parallelism modeling—increasingly important as more models adopt these features—which vTrain does not capture. Second, for a dense model that vTrain supports, we evaluate *configuration-search feasibility*, a criterion that measures how faithfully a tool captures system resource requirements. AIPerf prunes infeasible setups and returns the best feasible plan, whereas vTrain recommends a plan that in fact fails with out-of-memory (OOM). This underscores the importance of the detailed system-level resource modeling (e.g., memory) that AIPerf provides.

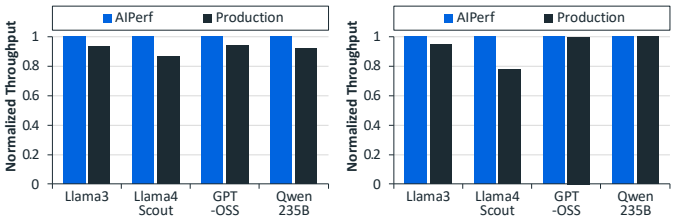
Even for feasible configurations, AIPerf achieves substantially better accuracy and fidelity than vTrain. For example, in configurations that apply selective activation recomputation in the backward pass to relieve memory pressure and that expose more inter-DP communication, AIPerf faithfully models the runtime behavior—partially recomputed activations and parameters, gradients, and optimizer states sharded across the network hierarchy—which vTrain underestimates. The gap widens further for TP8/PP1/DP8, where communication overhead becomes more exposed.

In production, incorporating these system- and model-level dynamics is key to projecting real training jobs, and these results show how effectively AIPerf captures them.



(a) H100 (b) GB300

Fig. 5: AIPerf headroom analysis for compute components.



(a) H100 (b) GB300

Fig. 6: AIPerf headroom analysis for network components.

IV. Case Studies

This section presents five case studies that span the co-design lifecycle, each highlighting a planning insight that is difficult to obtain without jointly modeling compute, memory, and communication across the full parallelism space: headroom analysis (§IV-A) and job planning (§IV-B) on deployed platforms, load-imbalance modeling for MoE architectures (§IV-C), and network topology (§IV-D) and scale-up domain size (§IV-E) for rack co-design.

A. Headroom Analysis

A performance model's prediction gap is usually treated as a limitation to minimize. In production, the opposite is true: the gap between projection and measurement *is the deliverable*—it tells the platform team exactly where optimization effort should go and how much throughput each fix is worth.

We demonstrate this on two platforms at different stages of maturity: H100 SXM (80 GB HBM, 8-GPU NVLink scale-up domain, ConnectX-7 400 Gbps RoCE scale-out, 700 W TDP), a mature platform optimized over multiple years, and GB300 NVL72 (288 GB HBM, 72-GPU NVLink scale-up domain, ConnectX-8 800 Gbps RoCE scale-out, 1200 W TDP), whose software stack is still being optimized. All experiments in this section use Meta's internally ported Megatron-LM framework [27]. Table IV summarizes the models, spanning dense and MoE architectures.

Figure 5 and Figure 6 compare AIPerf's per-layer projected throughput for computation and communication, respectively, against measured traces from Megatron-LM runs across all (model, platform) pairs. Projections closely match measurements: 5.6% median gap for computation and 8.5% for communication on H100, 9.4% and 7.5% on GB300,

Model	Method	Plan	TFLOP/sec
Llama-3	AIPerf	TP1/CP1/PP1/DP64	1225.9
	Practice A	TP4/CP1/PP4/DP4	1037.9
	Practice B	TP8/CP1/PP8/DP1	671.9
Qwen-3	AIPerf	TP2/CP2/PP1/DP16/EP32	376.9
	Practice A	TP2/CP2/PP2/DP8/EP8	331.8
	Practice B	TP8/CP1/PP1/DP8/EP8	228.1

TABLE V: Plan selection on GB300 (per-GPU throughput). AIPerf's exhaustive 5D search finds configurations that outperform expert-curated heuristics, demonstrating that locally sound parallelism choices can be globally suboptimal.

confirming that AIPerf captures system behavior well across both platforms.

Llama-4 Scout on GB300 is a notable outlier, with gaps of 14.5% for computation and 22.2% for communication, which are substantially higher than those of other models. This is where the gap becomes actionable: AIPerf's per-component breakdown traces it primarily to the overlapped execution of shared-expert and routed-expert GEMMs, a scheduling pattern unique to Llama-4 Scout's architecture, where resource contention and SM occupancy pressure degrade end-to-end throughput. This breakdown directly identifies an opportunity to recover the 14–22% gap in these components through improved scheduling of these overlapped operations, providing the kind of prioritized, quantitative target that replaces an undifferentiated backlog of performance work.

B. Job Planning

While headroom analysis identifies *where* to optimize, plan selection identifies *what* to run. Trial-and-error sweeps on live GPUs consume significant accelerator cycles, explore only a fraction of the 5D space, and must be repeated for every new model or sequence-length change. AIPerf replaces this with exhaustive enumeration. We demonstrate on GB300, showing how AIPerf finds configurations that expert-curated heuristics systematically miss.

Table V compares AIPerf's top-ranked configurations against expert-curated practice: parallelism plans selected by experienced engineers using common heuristics such as balancing across all parallelism degrees with model partitioning (Practice A) and setting TP to 8 to minimize communication overhead (Practice B). These heuristics are individually sound, yet AIPerf's exhaustive search finds configurations that outperform them. For Llama-3, AIPerf selects a pure data-parallel configuration that places the entire model on each rank, outperforming the best expert-curated plan by 1.18×. For Qwen-3, AIPerf balances across all parallelism dimensions, achieving 1.14× speedup over the best expert practice. The winning plans succeed not because any single axis is better tuned, but because the 5D interaction surface is highly non-convex: a locally suboptimal choice on one axis can unlock a globally better configuration by enabling a more favorable mapping on others.

AIPerf Setting	AIPerf Selected Plan EP	TP	ETP	PP	CP	DP	Measured Throughput
Uniform routing	16	1	1	4	4	16	1.00×
Imbalance-aware	8	2	1	4	1	32	**1.30**×

TABLE VI: AIPerf plan selection under uniform vs. imbalance-aware routing for GPT-OSS 120B on 256 H100 GPUs (MBS=1 for both), when routing imbalance exists.

Beyond parallelism degrees, AIPerf also optimizes how parallel groups are mapped onto the network hierarchy. For Qwen-3 on 64 GB300 GPUs, AIPerf selects TP2/CP2/PP1/DP16/EP32 as the best configuration, but the placement of these groups across the scale-up domain matters as well. One option concentrates DP groups within the scale-up domain, favoring Fully Sharded Data Parallel (FSDP)-related collectives; the alternative distributes DP shards across scale-up domains, favoring TP and CP collectives that benefit from high-bandwidth links. AIPerf identifies the latter placement (i.e., DP-distributed) as optimal, achieving 376.9 TFLOP/sec on the target cluster, a 1.81× speedup over the DP-concentrated mapping (208.3 TFLOP/sec).

These results demonstrate that expert intuition, while reasonable in low-dimensional settings, systematically underexplores the configuration space when five parallelism dimensions and their topological placement interact non-linearly.

C. Workload Imbalance

This section demonstrates how workload imbalance modeling (§III-C4) affects plan selection, a factor that prior planners either ignore or model in isolation from other parallelism decisions. We use GPT-OSS 120B on 256 H100 GPUs (32 nodes), whose 128 experts and top-4 routing produce AllToAll traffic with measurable routing imbalance, making it representative of production MoE workloads. We train on a mixture of open-source datasets spanning web text [28], Wikipedia, mathematics [29], code [30], books [31], scientific papers [32], and multilingual text [33].

We run AIPerf's 5D enumeration twice: once assuming uniform routing and once with the imbalance-aware modeling based on empirical characterization. Under uniform routing, AIPerf selects EP=16, TP=1 as the highest-throughput plan. With imbalance-aware routing, the best plan switches to EP=8, TP=2, which trades EP for TP. The main reason is that a smaller EP leads to less load imbalance and AIPerf projects that the benefit of reduced imbalance outweighs the overhead of TP communication.

To validate this decision, we ran both AIPerf-selected plans on the GPU cluster with the open-source dataset, which exhibits real routing imbalance. The imbalance-aware plan achieves 1.30× higher throughput than the uniform-routing plan (Table VI). Ignoring routing imbalance does not merely produce optimistic projections but selects a materially worse configuration, making imbalance-aware planning essential for MoE workloads at scale.

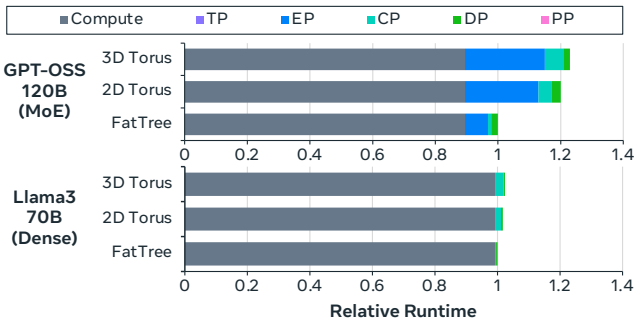


Fig. 7: Topology comparison (fat-tree, 2D torus, 3D torus) on a 4K MTIA 400 cluster. Iteration time is normalized to fat-tree. AIPerf selects the best 5D plan independently per topology.

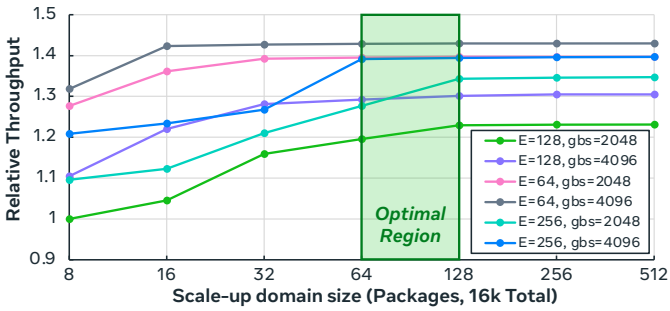


Fig. 8: Scale-up domain size analysis for Qwen-3 235B variants on a 16K MTIA 400 cluster.

The same holds for hardware decisions: ignoring expert imbalance overstates the benefit of larger EP degrees, and hence of larger scale-up domains (§IV-E).

D. Network Topology Exploration

Network topology choices must be evaluated against end-to-end performance across many models and parallelism settings, not against link-level bandwidth alone. We used AIPerf to compare three canonical scale-up topologies for an internal accelerator design: fat-tree, 2D torus, and 3D torus as scale-up network plus a common scale-out network. The per-accelerator scale-up and scale-out injection bandwidths are identical across all hardware configurations. The scale-up domain size is 64 for fat-tree, while the torus networks allow dynamic configurations for up to 4K accelerators.

As an example, Figure 7 compares the performance for GPT-OSS 120B (MoE, 128 experts, Top-4) and dense Llama-3 70B models for 4K MTIA 400 accelerators. As expected, the performance difference was minimal for Llama-3 because the dense model allowed hiding communication overheads behind compute. However, for the GPT-OSS 120B model, fat-tree showed a $\sim$20% runtime reduction because it has higher bisection bandwidth than torus networks, leading to significantly lower EP all-to-all communication time. For this model, we did not observe a significant difference between 2D and 3D torus, mainly because the model did not require full 5D parallelism.

Although we have shown results for two models above for illustration, in reality we need to evaluate a large sweep of model parameters to consider how models can change in the future. In addition, performance is only one of the factors in hardware decisions, and various other factors, including power and cost, need to be taken into account for the final decision.

E. Scale-up Domain Exploration

Scale-up network bandwidth is typically an order of magnitude higher than scale-out bandwidth on AI accelerator platforms. Increasing the scale-up domain size allows high-volume collectives to remain on the fast scale-up network, reducing communication overhead. Larger domains, however, raise rack power, cost, and hardware complexity while reducing reliability, so the throughput gain must be weighed against all four. AIPerf quantifies that gain across a wide range of AI models to inform the decision.

To illustrate how AIPerf is used for this critical decision, we describe how we applied it to the MTIA 400 rack design [2]. MTIA 400 is Meta's specialized chip designed to serve LLM workloads. Each accelerator provides 3 PFLOP/sec of BF16 performance and 288 GB of HBM at 9.2 TB/s bandwidth. Our MTIA rack connects MTIA 400 accelerators through two distinct networks: 1.2 TB/s scale-up bandwidth and 100 GB/s scale-out bandwidth, making the scale-up network 12$\times$ faster than scale-out.

Before finalizing the MTIA 400 rack design, we conducted a performance analysis for different domain sizes across our target workload mix. Figure 8 presents a representative example, sweeping the domain size from 8 to 512 MTIA 400 accelerators on a 16K MTIA 400 cluster with a two-level network hierarchy. The total accelerator count is held fixed; only the domain size varies. We evaluate variants of Qwen-3 235B with sequence length of 32768, different expert counts (64–256) and global batch sizes (2048, 4096). Our internal analysis included a much larger sweep, focusing on internal model trajectories, but the figure provided here is representative enough to demonstrate the methodology and the conclusions.

Despite the large network bandwidth disparity, throughput saturates at a scale-up domain size of just 64–128 packages. At this domain size, the HBM capacity is sufficient to support parallelism configurations that achieve high MFU with almost all high-volume communication traffic contained within the scale-up domain. Beyond this point, scale-out communication is no longer a performance bottleneck and doubling the domain size increases rack costs without meaningful throughput gains. Therefore, we recommended the 64–128 package region as the throughput sweet spot. The final rack design of MTIA 400 settled on 72 packages, balancing throughput against power, costs, reliability, and hardware constraints [2].

V. Related Work

We situate AIPerf among representative prior tools, none of which was designed to jointly satisfy the five requirements

(§II-C) that a multi-phase co-design lifecycle demands.

Analytical planners. Piper [3] and AMPeD [4] formulate parallelism selection over DP, TP, and PP with memory constraints, providing fast search but covering only three of five dimensions and omitting recomputation and offload costs (R1) and workload non-uniformity (R4). Calculon [5] adds schedule-aware timing with overlap-aware collective placement (R2) for dense transformers but does not extend to full 5D search (R1) or workload calibration (R4). Kundu et al. [34] provide workload-driven cost modeling but similarly omit 5D joint search (R1) and non-uniformity calibration (R4). Piper, AMPeD, and Kundu et al. further lack modular extensibility (R3) and cross-platform portability (R5). Calculon partially addresses both but remains limited to dense architectures.

Simulation- and trace-driven tools. Proteus [6] simulates execution at the operator level with schedule-aware timing (R2) and a pluggable topology model that partially addresses extensibility (R3), but relies on profiling traces that must be regenerated per platform, limiting cross-platform portability (R5). vTrain [7] is a closely related profiling-driven simulator that searches 3D parallelism (DP/TP/PP) with GPipe and 1F1B scheduling to find cost-effective LLM training plans. It models neither MoE & expert parallelism [14] nor context parallelism (CP), which shards long sequences to fit activation memory [35]; both are crucial for modern training, so its plan space is narrower than the 5D search (R1), and it does not calibrate for workload non-uniformity (R4). Table III quantifies the gap. DayDream [36] estimates the efficacy of training optimizations from a single-GPU dependency graph; predating LLM-scale parallelism, it does not address 5D plan search (R1) or workload non-uniformity (R4). Lumos [37] provides graph-based pipeline simulation (R2) but lacks topology-aware placement and modular extensibility (R3), and does not model workload non-uniformity (R4). These approaches achieve higher single-platform fidelity at the cost of the search scalability and cross-platform portability (R5) that production planning demands.

Cross-stack co-design frameworks. DeepFlow [8] models the full stack with topology what-if reasoning that partially addresses extensibility (R3), but uses simplified schedule modeling (R2) and does not jointly search 5D parallelism with memory-feasibility validation (R1) or calibrate for workload non-uniformity (R4).

No prior tool combines R1–R5, yet the combination is what makes cross-platform comparison trustworthy and cross-phase feedback possible. AIPerf addresses these gaps jointly (§III).

VI. Conclusion

We described how Meta's hardware–AI co-design lifecycle relies on a production performance planner spanning six phases of a multi-year effort. We derived five requirements that such a planner must satisfy jointly and showed how AIPerf addresses them. Our case studies demonstrate a consistent finding: jointly modeling the interacting features of distributed training—parallelism, scheduling, memory, topology, and workload non-uniformity—changes which decision is made, not merely the projected throughput. The planning insights that shaped real hardware and model decisions emerged only when the planner reasoned across all of these simultaneously. AIPerf has served this lifecycle for multiple years, informing training configurations for frontier models as well as MTIA rack architecture decisions that are now carried into fabrication.

References

[1] J. Coburn, C. Tang, S. Abu Asal, N. Agrawal, R. Chinta, H. Dixit, B. Dodds, S. Dwarakapuram, A. Firoozshahian, C. Gao, K. Gondkar, T. Graf, J. Hu, J. Huang, S. Hughes, A. Hutchin, B. Jakka, G. J. Chen, I. Kalyanaraman, A. Kamath, P. Kansal, E. Kazi, R. Levenstein, M. Maddury, A. Mastro, S. Medaiyese, P. Modi, J. Montgomery, S. Nadathur, A. Nagpal, A. Narasimha, M. Naumov, E. Ozer, J. Park, P. Ramani, H. Reddy, D. Reiss, D. Roy, S. Sekar, A. Sharma, P. Shetty, A. Sukumaran-Rajam, E. Tal, M. Tsai, S. Varshini, R. Wareing, O. Wu, X. Xie, J. Yang, H. Yu, T. Zargar, Z. Zeng, F. Zhang, A. Matthews, X. Jiao, J. Zhang, E. Menage, T. E. Stokke, and M. Sourouri, "Meta's second generation AI chip: Model-chip co-design and productionization experiences," in *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*, 2025.

[2] Meta AI, "Four MTIA chips in two years: Scaling AI experiences for billions," https://ai.meta.com/blog/meta-mtia-scale-ai-chips-for-billions/, 2026.

[3] J. M. Tarnawski, D. Narayanan, and A. Phanishayee, "Piper: Multidimensional planner for DNN parallelization," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/d01eeca8b24321cd2fe89dd85b9beb51-Paper.pdf

[4] D. Moolchandani, J. Kundu, F. Ruelens, P. Vrancx, T. Evenblij, and M. Perumkunnil, "AMPeD: An analytical model for performance in distributed training of transformers," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023.

[5] M. Isaev, N. McDonald, L. Dennison, and R. Vuduc, "Calculon: a methodology and tool for high-level co-design of systems and large language models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23)*, 2023.

[6] J. Duan, X. Li, P. Xu, X. Zhang, S. Yan, Y. Liang, and D. Lin, "Proteus: Simulating the performance of distributed DNN training," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[7] J. Bang, Y. Choi, M. Kim, Y. Kim, and M. Rhu, "vTrain: A simulation framework for evaluating cost-effective and compute-optimal large language model training," in *2024 IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024.

[8] N. Ardalani, S. Pal, and P. Gupta, "DeepFlow: A cross-stack pathfinding framework for distributed AI systems," *ACM Transactions on Design Automation of Electronic Systems*, 2024.

[9] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016. [Online]. Available: https://arxiv.org/abs/1604.06174

[10] V. A. Korthikanti, J. Casper, S. Lym, L. McAfee, M. Andersch, M. Shoeybi, and B. Catanzaro, "Reducing activation recomputation in large transformer models," in *Proceedings of Machine Learning and Systems (MLSys)*, 2023. [Online]. Available: https://arxiv.org/abs/2205.05198

[11] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "ZeRO: Memory optimizations toward training trillion parameter models," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '20)*, 2020.

[12] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "ZeRO-Offload: Democratizing billion-scale model training," in *Proceedings of the 2021 USENIX Annual Technical Conference (USENIX ATC '21)*, 2021. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/ren-jie

[13] P. Qi, X. Wan, G. Huang, and M. Lin, "Zero bubble pipeline parallelism," in *International Conference on Learning Representations (ICLR)*, 2024. [Online]. Available: https://arxiv.org/abs/2401.10241

[14] DeepSeek-AI, "DeepSeek-V3 technical report," *arXiv preprint arXiv:2412.19437*, 2024. [Online]. Available: https://arxiv.org/abs/2412.19437

[15] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[16] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: Generalized pipeline parallelism for DNN training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[17] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. A. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters using Megatron-LM," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*, 2021.

[18] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, 2022. [Online]. Available: https://jmlr.org/papers/v23/21-0998.html

[19] Meta Inference Team, "Optimizing deployment configurations for LLM inference," in *Ninth Conference on Machine Learning and Systems*, 2026.

[20] Y. Zhao, A. Gu, R. Varma, L. Luo, C.-C. Huang, M. Xu, L. Wright, H. Shojanazeri, M. Ott, S. Shleifer, A. Desmaison, C. Balioglu, P. Damania, B. Nguyen, G. Chauhan, Y. Hao, A. Mathews, and S. Li, "PyTorch FSDP: Experiences on scaling fully sharded data parallel," *Proceedings of the VLDB Endowment*, 2023.

[21] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. [Online]. Available: https://arxiv.org/abs/2205.14135

[22] T. Dao, "FlashAttention-2: Faster attention with better parallelism and work partitioning," in *International Conference on Learning Representations (ICLR)*, 2024. [Online]. Available: https://arxiv.org/abs/2307.08691

[23] A. Grattafiori *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[24] Meta AI, "The llama 4 herd of models," https://ai.meta.com/blog/llama-4-multimodal-intelligence/, 2025.

[25] "GPT-OSS 120b," https://huggingface.co/openai/gpt-oss-120b, 2025.

[26] Qwen Team, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025. [Online]. Available: https://arxiv.org/abs/2505.09388

[27] "NeMo Megatron Bridge," https://github.com/NVIDIA-NeMo/Megatron-Bridge, 2025.

[28] G. Penedo, H. Kydlíček, L. Ben Allal, A. Lozhkov, M. Mitchell, C. Raffel, L. Von Werra, and T. Wolf, "The FineWeb datasets: Decanting the web for the finest text data at scale," *arXiv preprint arXiv:2406.17557*, 2024. [Online]. Available: https://arxiv.org/abs/2406.17557

[29] K. Paster, M. D. Santos, Z. Azerbayev, and J. Ba, "OpenWebMath: An open dataset of high-quality mathematical web text," *arXiv preprint arXiv:2310.06786*, 2023. [Online]. Available: https://arxiv.org/abs/2310.06786

[30] D. Kocetkov, R. Li, L. B. Allal, J. Li, C. Mou, C. Muñoz Ferrandis, Y. Jernite, M. Mitchell, S. Hughes, T. Wolf, D. Baez, J. Jernigan, and L. von Werra, "The Stack: 3 TB of permissively licensed source code," *Transactions on Machine Learning Research*, 2022. [Online]. Available: https://arxiv.org/abs/2211.15533

[31] J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap, "Compressive transformers for long-range sequence modelling," *arXiv preprint arXiv:1911.05507*, 2019. [Online]. Available: https://arxiv.org/abs/1911.05507

[32] L. Soldaini and K. Lo, "peS2o (Pretraining Efficiently on S2ORC) Dataset," https://github.com/allenai/pes2o, 2023, oDC-By, Allen Institute for AI.

[33] S. Singh, F. Vargus, D. Dsouza, B. F. Karlsson, A. Mahendiran, W.-Y. Ko, H. Shandilya, J. Patel, D. Mataciunas, L. OMahony, M. Zhang, R. Hettiarachchi, J. Wilson, M. Machado, L. Souza Moura, D. Krzemiński, H. Fadaei, I. Ergün, I. Okoh, A. Alaagib, O. Mudannayake, Z. Alyafeai, V. M. Chien, S. Ruder, S. Guthikonda, E. A. Alghamdi, S. Gehrmann, N. Muennighoff, M. Bartolo, J. Kreutzer, A. Üstün, M. Fadaee, and S. Hooker, "Aya dataset: An open-access collection for multilingual instruction tuning," *arXiv preprint arXiv:2402.06619*, 2024. [Online]. Available: https://arxiv.org/abs/2402.06619

[34] J. Kundu, W. Guo, A. BanaGozar, U. De Alwis, S. Sengupta, P. Gupta, and A. Mallik, "Performance modeling and workload analysis of distributed large language model training and inference," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*, 2024.

[35] S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, S. L. Song, S. Rajbhandari, and Y. He, "DeepSpeed Ulysses: System optimizations for enabling training of extreme long sequence transformer models," *arXiv preprint arXiv:2309.14509*, 2023. [Online]. Available: https://arxiv.org/abs/2309.14509

[36] H. Zhu, A. Phanishayee, and G. Pekhimenko, "Daydream: Accurately estimating the efficacy of optimizations for DNN training," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 337–352.

[37] M. Liang, H. T. Kassa, W. Fu, B. Coutinho, L. Feng, and C. Delimitrou, "Lumos: Efficient performance modeling and estimation for large-scale LLM training," in *Proceedings of Machine Learning and Systems (MLSys)*, 2025. [Online]. Available: https://arxiv.org/abs/2504.09307