MLSynth: Towards Synthetic ML Traces

Adel Sefiane
Imperial College London
London, UK
NVIDIA
Roskilde, Denmark

Alireza Farshin
NVIDIA
Stockholm, Sweden

Marios Kogias
Imperial College London
London, UK

Abstract

AI infrastructure continues to grow rapidly to meet the escalating demand for compute power required to train and inference increasingly capable models. This growth brings significant challenges in both the design and operation of ML pipelines. Exploring these challenges and evaluating potential solutions can be prohibitively expensive and time-consuming without effective simulation tools. This paper introduces `MLSynth`, a framework for synthesising ML workloads, which is essential for meaningful benchmarking of AI infrastructure. More specifically, `MLSynth` allows researchers to: (*i*) define a wide range of ML models with different parallelisation strategies, (*ii*) explore various sources of performance variability, and (*iii*) generate synthetic Chakra execution traces that can be used with existing simulation frameworks (*e.g.,* ASTRA-Sim) to comprehensively model ML workloads.

CCS Concepts

• **Networks** → **Network simulations**; **Data center networks**; • **Computing methodologies** → Distributed artificial intelligence; **Distributed algorithms**.

Keywords

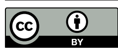
Synthetic Chakra Execution Traces, AI Training, Simulation, Large-Scale Clusters.

ACM Reference Format:
Adel Sefiane, Alireza Farshin, and Marios Kogias. 2025. `MLSynth`: Towards Synthetic ML Traces. In *2nd Workshop on Networks for AI Computing (NAIC '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3748273.3749211

1 Introduction

The Artificial Intelligence (AI) landscape has undergone a profound transformation since the emergence of ChatGPT and Large Language Models (LLMs). These innovations have catalysed an unprecedented proliferation of AI applications across domains. This rapid expansion has generated increasing demands for more sophisticated AI models capable of delivering highly accurate responses through larger architectures, enhanced reasoning capabilities, and multi-modal support that can process text, image, audio, and video.

This evolution in AI capabilities has precipitated an extraordinary surge in computational requirements and data centre capacity.



This work is licensed under a Creative Commons Attribution 4.0 International License.
NAIC '25, Coimbra, Portugal
© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2082-6/2025/09
https://doi.org/10.1145/3748273.3749211

Table 1: `MLSynth` offers an accurate, reproducible, and tunable way to generate ML workloads.

	Accurate?	Reproducible?	Tunable?
Real test-bed	✓	✗	✗
Non-AI workload	✗	✓	✓
Real Chakra ET	✓	✗	✗
SimAI AICB	✓	✓	✗
MLSynth	✓	✓	✓

For instance, training frontier models like GPT-4 requires thousands of high-performance GPUs operating continuously for weeks or months [13]. Consequently, it has become imperative to not only enhance training algorithms but also to design highly efficient "AI factories" – specialised clusters or data centres. The optimisation of these facilities presents a multifaceted challenge spanning scheduling algorithms [14], network topology designs [19], and congestion control [15], all of which must be calibrated according to specific cluster characteristics and operational requirements (*i.e.,* deployed models and workload).

Due to the prohibitive cost of infrastructure, end-to-end simulation of ML workloads is the only *cost-effective way* for innovation and exploring various trade-offs in designing AI infrastructure. Both academic research and industry development require accurate simulation capabilities; researchers to explore novel solutions, and industry practitioners to validate network configurations before committing substantial resources to physical deployment.

However, despite this necessity, a significant gap exists between network simulation and ML simulation approaches. ML-focused research [14, 15, 19] typically evaluates performance on physical test-beds, that are prohibitively expensive and/or rigid to evolve fast, or abstract network settings through simplified analytical models. A recent Meta study [3] revealed that even existing networking protocols such as DCQCN [23] remain barely explored in real ML deployments. Meanwhile, data centre networking-focused papers frequently neglect to incorporate realistic ML workload representation when evaluating their solutions [2, 7, 11], even when specifically motivated by the applications for AI. With the majority of data centre expansion serving the use-cases of AI training & inference, evaluating new contributions with accurate workloads has become increasingly relevant.

To bridge this gap, we present `MLSynth`, an open-source tool that generates accurate ML workloads as a Chakra Execution Trace (ET) [18] that represents the workload as a directed acyclic graph that can be used with full-stack simulators (*e.g.,* ASTRA-sim [21] + ns-3, Multiverse [5]) to enable clear correlations between high-level model parametrisation and low-level network layer metrics. `MLSynth` can define workloads by (*i*) high-level ML parameters

(*e.g.,* batch size, num layers, hidden size) and (*ii*) parallelisation strategies (*i.e.,* data, tensor, and pipeline parallelism). Furthermore, `MLSynth` enables researchers to induce tail events in the workload to observe the cascading impacts on every part of the workload *and* better understand performance variability of various components, which is too important to ignore in large deployments. For instance, a recent study from ByteDance [9] reveals that approximately 10.4% of allocated GPU hours wasted due to stragglers and 42.5% of training jobs experienced at least 10% slowdown in large-scale deployments. These performance variations emerge from diverse sources spanning both hardware and software domains, including CPU contention from co-located jobs, thermal throttling causing GPU frequency reduction, network-related issues, and systematic issues in workload partitioning via parallelisation strategies [4, 6].

We believe `MLSynth` is the first to: (*i*) generate fully synthetic ML workloads using the standardised Chakra format, and (*ii*) systematically enable the exploration of various sources of performance variability in AI infrastructure (*e.g.,* straggler GPUs & and NICs). `MLSynth` is publicly available at: github.com/NetMLSim/MLSynth. It includes the implementation for some baseline models (*i.e.,* Transformer & Mixture-of-Experts) and parallelisation strategies.

2 Existing Solutions

This section elaborates on existing solutions to generate ML workloads; see Table 1 for comparison.

Real world test-beds. Research in distributed machine learning infrastructure frequently relies on real-world test-beds for empirical evaluation, where physical infrastructure comprising dozens to hundreds of high-performance GPUs connected through sophisticated networking topologies serves as the experimental platform. Cassini [14], for instance, utilises a 24-server test-bed, each equipped with an A100 GPU and a NVIDIA ConnectX-5 RDMA NIC with 50-Gbps capacity, to demonstrate the benefits of network-aware job scheduling in ML clusters. Similarly, MLTCP [15] employs a 12-server test-bed with A100 GPUs and 50-Gbps RDMA NICs to validate their congestion control modifications. These physical environments provide the gold standard for accuracy as they capture the true behaviour of complex systems including hardware idiosyncrasies, operating system interactions, and the unpredictability of real-world conditions.

Despite their accuracy advantages, real-world test-beds present significant barriers to reproducibility and accessibility, as the substantial financial investment required, often millions of dollars for GPU clusters, limits such experimentation to *only* well-funded industrial research labs and a handful of academic institutions. Furthermore, even with identical hardware, reproducing experiments becomes challenging due to variations in system configurations, firmware versions, and environmental factors. *This lack of reproducibility* hampers research progress by preventing independent verification of results. Additionally, the obscurity of physical systems makes it difficult to isolate and understand specific phenomena, particularly with respect to network behaviour and performance anomalies.

Analytical workloads. In response to the limitations of real test-beds, researchers often resort to abstract workload simulations that approximate the communication patterns and computational demands of distributed ML training. These approaches typically employ flow-level simulations or analytical models that capture high-level traffic patterns such as all-reduce, all-to-all, or parameter server communication without modelling packet-level dynamics. Such abstractions offer practical advantages; they require minimal computational resources, provide quick time-to-solution, and enable exploration of large design spaces that would be prohibitively expensive with real hardware.

However, these abstract approaches are *not accurate enough* for precise infrastructure design, as they fail to capture critical network phenomena such as queuing dynamics, congestion control behaviour, and buffer management that significantly impact performance, particularly in tail events. As demonstrated in recent research, flow-level simulators systematically underestimate flow completion times, especially for short flows and at the tail [8]. Furthermore, the disconnect between these simplified models and the actual execution of ML workloads prevents insights into how higher-layer design decisions, such as scheduling strategies and parallelisation techniques, influence network traffic patterns. This separation obscures the complex interactions between computation and communication in ML training, where anomalies in one domain can affect the other, causing performance degradation.

Workload Traces. Accurate workload traces can be recorded from real test-beds and then used to replay the workload in a simulation. Simulators such as ASTRA-Sim [21] and Multiverse [5] make use of these kinds of traces, namely, Chakra ETs [18].

Chakra ETs represent a significant advancement towards standardised workload representation in machine learning simulation. Developed under the MLCommons initiative, Chakra provides an open, interoperable graph-based schema that captures key operations-including compute, memory access, and communication-along with their dependencies, timing constraints, and resource requirements. These execution traces effectively serialise the execution of distributed ML training into a comprehensive, framework-agnostic representation that can be shared, analysed, and simulated. When coupled with full-stack simulators, Chakra traces enable detailed modelling of both computation and communication aspects of ML workloads. This approach bridges the gap between abstract simulation and real-world testing by providing accurate workload representations while enabling controlled experimentation across varying system configurations.

Despite these advantages, the current Chakra ecosystem faces substantial limitations. Foremost among these is the continued reliance on real test-beds for trace collection. As noted by Meta researchers, "*Because traces are gathered from actual machine runs, the kernels executed are optimised for the specific system at play*" [1]. This dependency means that organisations without access to substantial hardware resources cannot generate traces representing their workloads of interest. Additionally, traces collected from specific hardware configurations inherit the topological constraints and parallelisation strategies employed during collection, limiting their flexibility for exploring alternative designs. For instance, a trace collected from a system using data parallelism cannot easily be adapted to evaluate pipeline parallelism or hybrid strategies. This inflexibility restricts the utility of collected traces for exploring the design space of emerging ML infrastructure. Furthermore, performance anomalies or tail events captured in the

trace may reflect peculiarities of the original test system rather than inherent characteristics of the workload, reducing transparency and potentially obscuring insights valuable for system optimisation.

Alternatively, a workload trace can be generated from scratch. By defining model parameters and parallelisation strategies, the workload can be recreated. To our knowledge, SimAI [20] is the closest existing approach to addressing the challenge of end-to-end ML infrastructure simulation. It is a very recent contribution, highlighting the active need across the research community. At its core, SimAI's AICB (AI Communication Benchmark) component offers a sophisticated method for generating representative workloads by mocking NCCL calls during a simulated training process. This approach avoids the need for physical GPU infrastructure while maintaining high fidelity to the communication patterns of real ML training workloads.

While SimAI AICB represents a significant advancement in generating high-fidelity workload traces without physical hardware, several critical limitations constrain its utility for comprehensive ML infrastructure research. The reproducibility of AICB-generated traces remains problematic due to their strong dependency on specific NCCL implementation, which introduces subtle environmental shifts across different research settings. This makes reproducibility more difficult, and renders the workload more opaque as a whole. Furthermore, despite the association between generated traces and high-level model parameters, AICB suffers from *limited tunability*. The complex sequence of kernels produced, combined with its non-standard output format, obscures workload characteristics and impedes researchers' ability to introduce purposeful variations for controlled experimentation. This lack of malleability makes it difficult to isolate specific phenomena or test hypothetical scenarios, such as straggler GPUs and faulty equipment, that could yield insights into system behaviour under varied conditions. Finally, the AICB workload generator lacks a modular framework for integrating new architectures and parallelisation strategies. The system is built around a fixed set of predetermined frameworks with script-based interfaces that require users to work within predefined model configurations and parameter spaces. The absence of well-defined component interfaces makes it difficult to introduce novel neural network architectures or emerging parallelisation paradigms, constraining its utility to the specific benchmarking scenarios it was originally designed to support rather than enabling flexible exploration of diverse ML infrastructure configurations.

3 `MLSynth`: Synthesise ML Workloads

`MLSynth` introduces a framework for synthesising ML workloads, which addresses the limitation of existing solutions (see Table 1). Users can use `MLSynth` to define high-level ML model configurations and then generate Chakra ETs. Chakra ETs serve as the critical intermediary representation, providing a standardised schema that simplifies the exchange of workload information across different simulation frameworks. By generating these traces synthetically rather than capturing them from real systems, `MLSynth` overcomes the limitations of physical test-bed requirements, maintains accuracy & reproducibility, and enables more exploration & experimentation capabilities. For example, `MLSynth` enables workload scaling, which is a useful method for scaling down

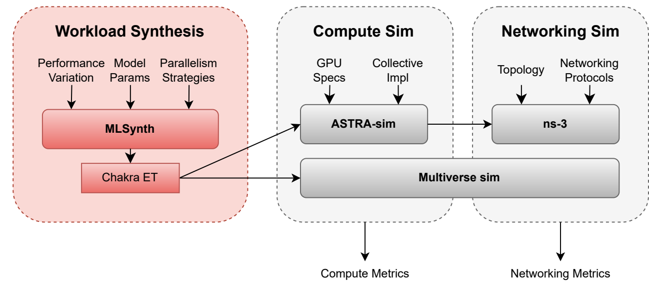


Figure 1: End-to-end workflow enabled by `MLSynth`.

experiments to shorten simulation time and explore a larger space. `MLSynth` supports workload scaling across multiple dimensions, allowing researchers to explore how performance characteristics evolve with increasing model size, dataset size, or compute.

The synthesised traces capture core operations – including compute, memory, and communication – along with their dependencies, timing, and resource constraints. The generated traces are subsequently fed into any compatible computational simulators (*e.g.,* ASTRA-sim [21] or Multiverse [5]) which utilise ns-3 for detailed network simulation; see Figure 1 for the complete workflow. This integrated approach creates a direct correlation between high-level ML model configurations and low-level networking metrics, enabling researchers to evaluate infrastructure & protocol designs without requiring physical hardware.

3.1 Modular Design

We design `MLSynth` with a modular architecture to be able to model workloads through clearly separated components, each representing individual parts of the distributed training process. This design enables users to modify or swap out any component to adapt the workload representation, providing flexibility for exploring different model architectures and parallelisation strategies. Figure 2 illustrates the four main components in `MLSynth`.

① **Layer.** At the core lies the Layer template, which represents individual layers that compose neural network models. This class generates the directed graph of Chakra nodes for operations within a single layer, returning the compute operations performed during forward and backward passes. To support tensor parallelism, the Layer interface accepts configuration flags that modify both compute and communication operations as needed, enabling accurate modelling of distributed computation within layers.

② **Model.** The Model template represents complete ML models as sequences of layers. Since neural networks fundamentally consist of layer compositions, this class tracks layer progression and propagates compute nodes generated by each layer to higher-level components. The Model abstraction maintains the structural integrity of the neural network while providing a clean interface for workload generation.

③ **Orchestrator.** The Orchestrator component models parallelism strategies used to distribute computation across GPUs. It contains a Model instance and orchestrates which layers are computed by each GPU, inserting communication operations between compute nodes returned by the Model class.

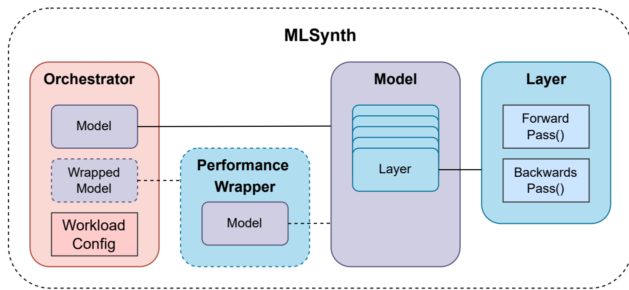


Figure 2: High-level design of MLSynth.

④ **Performance Wrapper.** Real-world deployments frequently experience hardware-induced performance fluctuations that current simulators typically neglect [9, 22]. A unique contribution of MLSynth is its capability to model realistic performance variations that affect production ML training environments. Rather than requiring modifications to simulation frameworks, MLSynth embeds these variations directly in the generated workload; making exploration easy and simulation-agnostic. Note that because MLSynth only provides the workload, it can fundamentally only model pre-determined performance variations; any real-time variation at the system or network level is modelled by the underlying simulation stack.

Because the workload is fully synthetic and thus completely transparent, any kind of modification to the generated Chakra nodes can be done. We model performance variation via wrapper objects (*i.e.,* Performance Wrappers) that intercept calls between the Model class and the Orchestrator class. When intercepting the calls, the performance wrapper modifies the directed graph of operations that are returned by the Model – whether this is changing FLOPs for compute nodes, or adding 'wait' nodes to delay some operations to simulate slowdowns. This design preserves the separation of concerns, allowing delay models to be applied without modifying the underlying model implementations.

MLSynth's components can be configured via a comprehensive parameter set shown in Table 2. These parameters fall into three primary categories: model architecture specifications, parallelisation strategies, and performance variance configuration.

3.2 Implementing a Workload

To synthesise a workload for a specific model in MLSynth, we need to define the model architecture using the Orchestrator, Model, and Layer components. Additionally, if the workload requires experimenting with various performance variations, one should implement Performance Wrappers to model relevant operations. Any architecture or parallelisation strategy can be modelled by implementing their respective components. Next, we elaborate the details of defining the Transformer architecture in MLSynth.

Layer. A standard transformer layer consists of a multi-head attention mechanism followed by a two-layer feed-forward network, each preceded by layer normalisation and succeeded by residual connections. For each component, we calculate floating-point operations (FLOPs) according to tensor dimensions and operations as specified in Megatron-LM [17]. Intermediate steps

Table 2: MLSynth configuration parameters.

Configuration	Description
Model Architecture	
num_gpus	Number of GPUs used to train
batch_size	Overall batch size
micro_batch_size	Size of micro-batches processed by each GPU
num_layers	Number of layers in the model
hidden_dim	Hidden size of model
vocab_size	Size of vocabulary used in the model
scale	Configures the scaling of the model
Parallelisation	
dp_size	Size of Data Parallel group
pp_size	Size of Pipeline Parallel group
tp_size	Size of Tensor Parallel group
Variance per GPU	
GPU Id	Id of the GPU being configured
GPU slowdown chance	Chance that compute slowdown is experienced at each layer
NIC slowdown chance	Chance that NIC slowdown is experienced at each layer
GPU layer slowdown	Which layers compute slowdown is experienced
NIC layer slowdown	Which layers NIC slowdown is experienced
GPU slowdown	Amount of compute slowdown
NIC slowdown	Amount of NIC slowdown

such as normalisation are excluded, as they are computationally negligible. For the attention mechanism, the FLOP count is derived from the sequence length, hidden dimension, and number of attention heads, accounting for the query-key-value projections, attention matrix computation, and output projection. Similarly, for the feed-forward network, we calculate FLOPs based on the hidden dimension and intermediate dimension, typically four times larger than the hidden dimension in standard implementations.

When tensor parallelism is employed, our implementation introduces additional communication operations that reflect the distributed computation strategy. Following the Megatron-LM approach, we implement a row-split strategy for the attention block and a column-split strategy for the MLP block. This parallelisation requires all-reduce collective operations after the computation in each block to synchronise the partial results across participating processors. The communication volume for these operations is directly proportional to the hidden dimension size, and the generated Chakra nodes accurately represent both the data dependencies and the communication topology of these operations.

Model. We implement the Transformer model using the individual layers, including specialised components such as the embedding layer that precedes the transformer blocks. The majority of compute in a transformer layer lies in the attention block and the feed-forward network block [12]. The embedding layer implements vocabulary projection and positional encoding, essential preprocessing steps before the core transformer computation begins. Furthermore, we also implement the Mixture-of-Experts (MoE) [16], which replaces the dense feed-forward sub-layer with a pool of E expert MLPs whose weights are not shared. A lightweight top-k gating network routes each token to k experts and blends their outputs. This reduces the amount of compute required and adds some randomness to the training process within the MoE layer.

Orchestrator. We implement all of the standard parallelisation strategies. Our provided implementation includes Data Parallelism, Tensor Parallelism, and Pipeline Parallelism. Of note, our implementation of pipeline parallelism includes the default GPipe and 1F1B solution as in Megatron-LM [17]. Expert parallelism has been

gaining traction as a clean way to shard layers such that compute is evenly distributed for MoE models, with recent works going as far as using it inter-node [10]. Therefore, we have also included it. In expert parallelism, the E experts of every layer are sharded across different groups. The probability of each token being assigned to an expert is modelled via a uniform distribution. This models the end-goal of load-balancing loss [16], which aims to distribute tokens evenly amongst all experts. The tokens are shared amongst all shards via an all-to-all, with the resulting output from each expert shared amongst all shards with another all-to-all.

Performance Variation. We implement two types of performance variations: (*i*) GPU delays and (*ii*) NIC delays. The GPU delays simulate phenomena such as thermal throttling, intra-die performance variations, power fluctuations, and component ageing by increasing the computational requirements (FLOPs) for specific nodes by adjusting their computation specifications. Similarly, NIC delays model network hardware disruptions by inserting communication pauses before specific transmission operations. These variations can be applied systematically or stochastically, with configurable distributions to represent different failure modes and their probabilities. This serves as an example of how different what-ifs can be modelled using `MLSynth`.

4 Evaluation

This section focuses on two main questions to validate `MLSynth`'s capabilities: (*i*) How accurate are the synthetic traces generated by `MLSynth` and (*ii*) How can `MLSynth` enable new types of analysis that were previously inaccessible. More specifically, we focus on reproducing the results from Megatron-LM [17].

Experiment setup. We use ASTRA-sim with the ns-3 back-end. Our simulation uses similar hardware characteristics to NVIDIA's Selene supercomputer DGX A100 nodes used in the original study, where 8 GPUs per node are connected via NVLink over which the tensor parallel communication occurs. As communication over NVLink has negligible congestion relative to internode congestion, we simulate it analytically so as to significantly reduce simulation time. The DGX A100 nodes in Selene are connected in a three-level fat tree. For this experiment only up to 8 × DGX A100 nodes are used (one for each pipeline stage). ASTRA-sim's default configuration simulates A100 GPUs at theoretical peak performance (312 TFLOPs) for FP16 precision. We calibrated our simulation by linearly scaling performance to 55% of theoretical maximum to align with empirically observed metrics from Megatron-LM.

4.1 Validating Accuracy

To assess `MLSynth`'s accuracy in generating representative workloads, we replicate a foundational experiment from the Megatron-LM [12] paper, which investigates the scaling efficiency of 1F1B (one-forward-one-backward) pipeline parallelism with varying batch sizes. This experiment represents a critical benchmark in distributed training research, as it isolates the impact of pipeline bubbles[1] on overall training throughput, which significantly influences the design decisions in large-scale training infrastructure.

[1] A bubble is the period during which certain GPUs remain idle due to dependencies in the pipeline execution schedule.

The experimental methodology maintains a constant tensor parallel group size of 8 while incrementally adding pipeline stages. Crucially, as each new pipeline stage is introduced, we proportionally scale the model size to ensure consistent computational load (FLOPs) per GPU. This approach specifically isolates the impact of pipeline parallelism on scaling efficiency by controlling for computational workload variability across configurations.

Figure 3 demonstrates that the *synthetic* workload generated by `MLSynth` successfully reproduces a key finding from the original Megatron-LM study *smaller batch sizes lead to diminished scaling efficiency as pipeline parallel dimensions increase.* This performance degradation occurs because smaller batch sizes result in fewer micro-batches available for pipeline scheduling, which consequently increases the relative size of the pipeline bubble. Regardless of absolute accuracy depending on the underlying system and network simulators, our results demonstrate that `MLSynth` generates workloads exhibiting scaling behaviours consistent with physical test-beds.

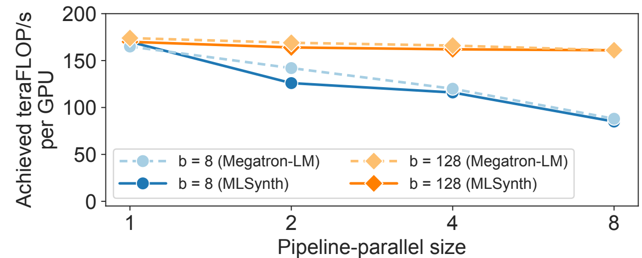


Figure 3: The effective teraFLOP/s over every GPU as pipeline size – and consequently, bubble size – increases. `MLSynth` successfully replicates Megatron-LM findings [17].

Scaling down traces. Simulating the experiment took progressively longer as we scaled the number of pipelines. We tested whether the simulation could be scaled down while maintaining accuracy. Scaling down the model to 1% by linearly decreasing the number of FLOPs, tensor size, and communication size decreased the runtime from 24 hours to 23 minutes for the most expensive setup of 8 pipeline stages with 128 batches. With the scaled down workload, we observed the same behaviour as with the full size workload – highlighting how a workload can easily be scaled to enable quick iteration while maintaining accuracy.

4.2 Performance Variation Analysis

Building on previous results, we demonstrate `MLSynth`'s capability to model realistic performance variability. This basic experiment highlights how `MLSynth` can be configured to induce tail events that significantly impact distributed training efficiency; an analysis that would previously require re-running the real test-bed with extensive observability. We leverage `MLSynth`'s performance variation features to introduce controlled anomalies, specifically modelling a scenario where a single GPU experiences a 10% computational slowdown during the processing of a single micro-batch, due to hardware-induced performance fluctuations such as thermal throttling or component ageing [9].

Our methodology enables precise control over the timing, magnitude, and location of performance variations by increasing the FLOP count for specific operations in the affected micro-batch. Figure 4 shows the results of our experiment. As expected, the slowdown extends the overall iteration time proportionally to the magnitude of the performance degradation. Scaling the setup up to 64 GPUs reveals a constant relationship between slowdown and training time, with the 21-ms increase in compute time resulting in training time increasing for 21 ms, no matter the scale of the experiment. Another observation concerns resource utilisation efficiency — while training time increases by approximately S seconds (where S represents the absolute slowdown duration), the cumulative wasted GPU time across the entire system increases by $S \times num_gpus$, as all GPUs must remain idle while waiting to synchronise communication with the straggler. This highlights how a single straggler can have a disproportionate effect, particularly in cloud environments where GPU resources are billed by time. While the delay in training time might appear modest, the economic cost of wasted GPU-hours becomes substantial at scale.

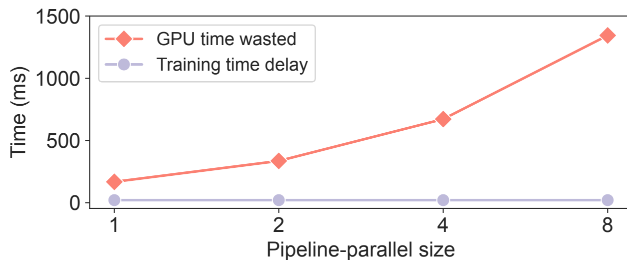


Figure 4: Using `MLSynth` to model how the impact of a single straggler scales. The wasted GPU time increases while end-to-end training time remains unaffected.

While this experiment demonstrates a straightforward performance degradation scenario, it exemplifies `MLSynth`'s broader capability to model complex system behaviours challenging to study in production environments. `MLSynth` can simulate controlled tail events, enabling the research of advanced straggler recovery mechanisms and sophisticated fault tolerance strategies. This enables research that would be prohibitively expensive to test on real hardware at scale, ultimately accelerating development of more resilient distributed ML training systems.

4.3 Flow Completion Time vs. Training Time

Finally, to illustrate the importance of full-stack analysis, we conduct an experiment demonstrating that Flow Completion Time (FCT) does not directly correlate with end-to-end training performance. This challenges the conventional use of FCT as the primary network optimisation metric. We configure a workload using `MLSynth` with GPipe parallelism, implementing synchronous data parallel all-reduce operations after each pipeline stage completes. The experiment uses the same model configuration from previous sections, organised into 4 pipeline groups and 4 data parallel groups. We employ GPipe without micro-batching to amplify the drain phase effects, making the analysis more apparent. We vary the effective bandwidth available to the all-reduce collective of the data parallel communication for the first pipeline stage. As shown in Figure 5, FCT increases linearly as bandwidth decreases. However, training time remains unaffected until the collective communication duration becomes the critical path starting around $50Gbps$ – specifically, when the all-reduce operation lasts until after the final pipeline stage finishes, thus holding back the entire training fleet. These intricacies become significantly more complex when employing sophisticated implementations of parallelism strategies such as inter-DC expert parallelism, DualPipe pipeline parallelism [10], and hybrid approaches that combine multiple parallelisation dimensions, where the interdependencies between communication patterns, computation scheduling, and resource utilisation create non-intuitive performance bottlenecks that traditional metrics fail to capture. This analysis demonstrates how `MLSynth` enables researchers to bridge the gap between individual system metrics and end-to-end performance, providing the transparency needed to understand these nuanced interactions in distributed ML training.

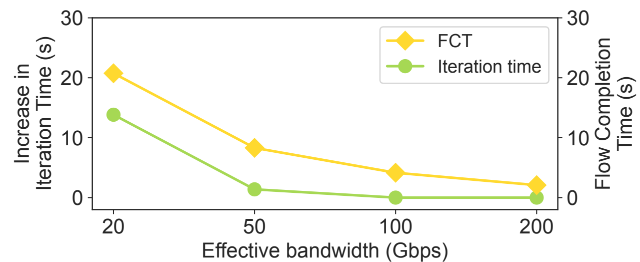


Figure 5: How FCT and iteration time increase as a data parallel communication step receives less bandwidth. An increase in FCT does *not* directly correlate with end-to-end training time.

5 Conclusion

We presented `MLSynth`, an open-source framework that enables researchers to synthesise ML workloads. By providing accurate, reproducible, and tunable workloads, `MLSynth` addresses a critical gap in AI infrastructure research. Our current implementation accurately simulates fundamental compute and communication patterns across various parallelisation strategies. We plan to extend `MLSynth` to (i) include memory access modelling and complex optimisations such as staggered communication and compute-communication overlap; (ii) synthesise inference workloads; and (iii) support multi-tenant scenarios.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments and suggestions on this paper. This work has been conducted with the support of the European Union EMPOWER-6G Doctoral Network under Grant Agreement No 101120332.

References

[1] 2023. Using Chakra execution traces for benchmarking and network performance optimization. https://engineering.fb.com/2023/09/07/networking-traffic/chakra-execution-traces-benchmarking-network-performance-optimization/

[2] Peirui Cao, Wenxue Cheng, Shizhen Zhao, and Yongqiang Xiong. 2024. Network Load Balancing with Parallel Flowlets for AI Training Clusters. In *Proceedings of the 2024 SIGCOMM Workshop on Networks for AI Computing (NAIC '24)*. Association for Computing Machinery, New York, NY, USA, 18–25. https://doi.org/10.1145/3672198.3673794

[3] Adithya Gangidi, Rui Miao, Shengbao Zheng, Sai Jayesh Bondu, Guilherme Goes, Hany Morsy, Rohit Puri, Mohammad Riftadi, Ashmitha Jeevaraj Shetty, Jingyi Yang, Shuqiang Zhang, Mikel Jimenez Fernandez, Shashidhar Gandham, and Hongyi Zeng. 2024. RDMA over Ethernet for Distributed Training at Meta Scale. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 57–70. https://doi.org/10.1145/3651890.3672233

[4] Aaron Grattafiori et al. 2024. The Llama 3 Herd of Models. https://doi.org/10.48550/arXiv.2407.21783 arXiv:2407.21783 [cs].

[5] Fei Gui, Kaihui Gao, Li Chen, Dan Li, Vincent Liu, Ran Zhang, Hongbing Yang, and Dian Xiong. 2025. Accelerating Design Space Exploration for {LLM} Training Systems with Multi-experiment Parallel Simulation. 473–488. https://www.usenix.org/conference/nsdi25/presentation/gui

[6] Tao He, Xue Li, Zhibin Wang, Kun Qian, Jingbo Xu, Wenyuan Yu, and Jingren Zhou. 2023. Unicron: Economizing Self-Healing LLM Training at Scale. https://doi.org/10.48550/arXiv.2401.00134 arXiv:2401.00134 [cs].

[7] Yanfang Le, Rong Pan, Peter Newman, Jeremias Blendin, Abdul Kabbani, Vipin Jain, Raghava Sivaramu, and Francis Matus. 2024. STrack: A Reliable Multipath Transport for AI/ML Clusters. https://doi.org/10.48550/arXiv.2407.15266 arXiv:2407.15266 [cs].

[8] Chenning Li, Anton A. Zabreyko, Arash Nasr-Esfahany, Kevin Zhao, Prateesh Goyal, Mohammad Alizadeh, and Thomas Anderson. 2025. m4: A Learned Flow-level Network Simulator. https://doi.org/10.48550/arXiv.2503.01770 arXiv:2503.01770 [cs].

[9] Jinkun Lin, Ziheng Jiang, Zuquan Song, Sida Zhao, Menghan Yu, Zhanghan Wang, Chenyuan Wang, Zuocheng Shi, Xiang Shi, Wei Jia, Zherui Liu, Shuguang Wang, Haibin Lin, Xin Liu, Aurojit Panda, and Jinyang Li. 2025. Understanding Stragglers in Large Model Training Using What-if Analysis. https://doi.org/10.48550/arXiv.2505.05713 arXiv:2505.05713 [cs].

[10] Aixin Liu et al. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[11] Rixin Liu, Menghao Zhang, Zihan Niu, Zili Meng, and Xiaohe Hu. [n. d.]. Themis: Efficiently Mitigating Congestion-Induced Fairness Disparities in Long-Haul RDMA Networks. ([n. d.]).

[12] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. https://doi.org/10.48550/arXiv.2104.04473 arXiv:2104.04473 [cs].

[13] OpenAI, Josh Achiam, et al. 2024. GPT-4 Technical Report. https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774 [cs].

[14] Sudarsanan Rajasekaran, Manya Ghobadi, and Aditya Akella. 2024. {CASSINI}: {Network-Aware} Job Scheduling in Machine Learning Clusters. 1403–1420. https://www.usenix.org/conference/nsdi24/presentation/rajasekaran

[15] Sudarsanan Rajasekaran, Sanjoli Narang, Anton A. Zabreyko, and Manya Ghobadi. 2024. MLTCP: A Distributed Technique to Approximate Centralized Flow Scheduling For Machine Learning. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks (HotNets '24)*. Association for Computing Machinery, New York, NY, USA, 167–176. https://doi.org/10.1145/3696348.3696878

[16] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv:1701.06538 [cs.LG] https://arxiv.org/abs/1701.06538

[17] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. https://doi.org/10.48550/arXiv.1909.08053 arXiv:1909.08053 [cs].

[18] Srinivas Sridharan, Taekyung Heo, Louis Feng, Zhaodong Wang, Matt Bergeron, Wenyin Fu, Shengbao Zheng, Brian Coutinho, Saeed Rashidi, Changhai Man, and Tushar Krishna. 2023. Chakra: Advancing Performance Benchmarking and Co-design using Standardized Execution Traces. https://doi.org/10.48550/arXiv.2305.14516 arXiv:2305.14516 [cs].

[19] Weiyang Wang, Manya Ghobadi, Kayvon Shakeri, Ying Zhang, and Naader Hasani. 2024. Rail-only: A Low-Cost High-Performance Network for Training LLMs with Trillion Parameters. In *2024 IEEE Symposium on High-Performance Interconnects (HOTI)*. 1–10. https://doi.org/10.1109/HOTI63208.2024.00013 ISSN: 2332-5569.

[20] Xizheng Wang, Qingxu Li, Yichi Xu, Gang Lu, Dan Li, Li Chen, Heyang Zhou, Linkang Zheng, Sen Zhang, Yikai Zhu, Yang Liu, Pengcheng Zhang, Kun Qian, Kunling He, Jiaqi Gao, Ennan Zhai, Dennis Cai, and Binzhang Fu. 2025. {SimAI}: Unifying Architecture Design and Performance Tuning for {Large-Scale} Large Language Model Training with Scalability and Precision. 541–558. https://www.usenix.org/conference/nsdi25/presentation/wang-xizheng-simai

[21] 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 *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 283–294. https://doi.org/10.1109/ISPASS57527.2023.00035

[22] Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Wu, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. 2024. FALCON: Pinpointing and Mitigating Stragglers for Large-Scale Hybrid-Parallel Training. https://doi.org/10.48550/arXiv.2410.12588 arXiv:2410.12588 [cs].

[23] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 523–536. https://doi.org/10.1145/2829988.2787484