MORPHLING: Emulator for Distributed Machine Learning at the Edge

Leyang Xue[†], Yufeng Xia[†], Eren Mendi[†], Ismaeel Bashir[†], Jiaxun Yang[†], Myungjin Lee[‡], Mahesh K. Marina[†]

[†]The University of Edinburgh; [‡]Cisco Research

Foundation models (FMs) are at the heart of the AI revolution, but training them is the exclusive domain of a handful of cloud providers. A growing line of work argues that *decentralised* FM training—pooling idle compute across volunteered edge devices such as laptops, tablets, and smartphones—can democratize the ecosystem. Modern consumer silicon makes this plausible: an Apple M4 Max delivers ∼38 TFLOPS, within an order of magnitude of a datacentre GPU, and edge devices spend most of their time idle. Realising this vision, however, requires evaluating training protocols, schedulers, and parallelism strategies at *realistic scale*: hundreds to thousands of heterogeneous devices, with thermal throttling, churn, network variability, and stragglers. Doing so on physical fleets is prohibitively expensive, operationally brittle, and hinders reproducibility.

Researchers naturally turn to emulation, particularly before deployment, but existing tools do not jointly satisfy the requirements of distributed edge-ML studies. Mobile device emulators such as Android Studio or VM-style systems faithfully reproduce guest stacks, yet they duplicate model and runtime state per device, depend on host schedulers for sharing, and saturate at small fleet sizes. Performance models predict accelerator latency or throughput, but they lack a distributed runtime that preserves framework interfaces, barrier semantics, compute–communication overlap, and straggler-induced tail latency. Federated-learning and distributed-training simulators scale further, but they usually omit device-level thermal dynamics, accelerator heterogeneity, and stride-sensitive compute effects. Credible evaluation therefore needs an emulator that combines *scalability*, *single-device fidelity*, and *distributed-device fidelity*.

We present MORPHLING[1], a measurement-driven emulator with a *unified execution backend* for real and emulated edge devices. MORPHLING runs unmodified PyTorch and JAX training scripts across mixed deployments in which a few physical devices can be combined with hundreds or thousands of virtual devices. The key idea is to intercept the stable dense-linear-algebra dispatch boundary used by modern ML frameworks—for example, `cblas_sgemm` in Apple Accelerate on iOS and LiteRT AOT-compiled kernels on Android—instead of virtualizing the full device software stack. Emulated devices execute the operations that the training script would issue, while a calibrated timing and thermal model supplies device-specific progress; the same broker, launcher, memory, and communication subsystems serve both real and virtual participants.

MORPHLING decouples tensor storage from emulated device instances via a centralized memory pool exposed through shared-memory handles. This removes the per-device model-replica duplication that makes naive emulation memory-bound, while remaining compatible with native CUDA contexts, pinned buffers, and checkpoint loading. The unit of execution is a task—a GEMM, GEMV, tensor transfer, or synchronization event—tagged with semantic and performance metadata, which lets the emulator preserve dependency ordering without requiring every virtual device to advance at host wall-clock speed. MORPHLING uses an event-driven *virtual timeline* in which each emulated device maintains independent compute and communication clocks. Compute and transfer events advance those clocks independently and meet only at explicit synchronization points; barriers advance to the slowest participant, reproducing straggler delay in data-parallel and federated training. This preserves the execution semantics of distributed training while allowing flexible variation of device mixes, network conditions, and parallelism strategies.

MORPHLING exploits the observation that transformer training is dominated by a small set of backend operations such as GEMM and GEMV. Its latency model is *stride-aware*: it keeps matrix-shape features rather than collapsing work into scalar FLOPs, because row- versus column-major traversal and tiling can change edge-accelerator cost by orders of magnitude. Its thermal model combines a Newton's-law-of-cooling backbone with an RBF correction over GEMM dimensions and sensor temperatures, capturing the smooth cooling regime as well as multi-stage DVFS throttling that a single time constant cannot represent.

Our prototype comprises about 5,000 lines of C++ and 1,500 lines of Python and supports unmodified user scripts. This implementation will be made open source shortly. The current dense-linear-algebra backend covers transformer workloads, with the same dispatch-layer interception and calibration procedure extending to other operator families. On mobile targets, we calibrate and validate against an iPhone 15 and a Samsung S24 Ultra using Apple Accelerate and LiteRT. The C++ dispatch interceptor adds 42.7 ns per call, below OpenBLAS scheduling jitter, and the Python hook adds 14.97 μs per call.

Evaluation shows that MORPHLING is both scalable and faithful. On a single 8×A5000 server with 1 TB host memory, MORPHLING scales to **1,800 concurrent emulated devices**, compared with 150 for a federated-learning-style simulator and 256 for Android-Studio-based emulation in the same testbed. With real GEMM from OPT and LLaMA-2, the performance model in MORPHLING is close to the real on-device measurement. End-to-end validation over 100 OPT training steps on real and mixed real/emulated deployments yields 5.8–11.6% MAPE, showing that MORPHLING preserves user-visible distributed behavior even when individual operator predictions vary.

[1]L. Xue et al., "Morphling: Emulator for distributed machine learning at the edge," in *Proceedings of the 24th Annual International Conference on Mobile Systems, Applications and Services Workshops*, ACM, 2026. DOI: 10.1145/3812836.3814779