
BITWISE-IDENTICAL MACHINE LEARNING ACROSS METAL, CUDA, AND HIP

Andrew Hendel¹

ABSTRACT

The same machine-learning workload can produce different bits on different GPUs, changing predictions, learned models and subsequent training updates. Mojolearn is a GPU machine-learning library whose default *identical* mode produces bitwise-identical results across verified Apple, NVIDIA and AMD GPUs and x86-64 and Arm CPUs. Results agree bit for bit, not merely within a numerical tolerance. It supports training and inference for over 100 machine-learning algorithms in 13 families, from decision trees to neural networks. Neural models trained in other frameworks can be served with identical outputs across vendors, and training workloads can be handed off between GPU vendors mid-run, or shared by GPUs of different vendors at once, while maintaining bitwise identity. Two training runs of a 162-million-parameter GPT-3 Small-shaped model follow different hardware routes collectively spanning Apple, NVIDIA and AMD GPUs and remain bitwise identical at all 5,000 optimizer steps, each processing 2.62 billion tokens. Neural training and inference run in full FP32 floating-point arithmetic, without reducing the computation to integers. To my knowledge, I am the first to enable these capabilities.

One Mojo codebase implements these algorithms across Metal, CUDA and HIP under a shared numerical contract. Mojo compiles the code for different GPUs, but the numerical contract, introduced in this work, fixes arithmetic order, rounding and elementary functions while allowing backend-specific scheduling.

1 INTRODUCTION

A model may serve predictions on NVIDIA GPUs, run locally on an Apple laptop, and later be evaluated on an AMD server. Even with identical weights and inputs, different GPU implementations can produce different output bits. For deployments that require exact replay, hardware becomes another variable in serving, regression testing and auditing. Mojolearn is a GPU machine-learning library whose default *identical* mode produces bitwise-identical outputs across verified devices and extends that guarantee to supported training computations. It covers over 100 algorithms in 13 families, from decision trees to neural networks, with Apple M2, M3 and M4, NVIDIA H100, A100, RTX 4090 and L40S, and AMD MI325X and MI300X GPUs and x86-64 and Arm CPUs in its verification scope. Unsupported dtypes, shapes or options raise an error rather than silently weakening the numerical guarantee. Section 3 states the conditions and limits of that guarantee.

¹Latent Archon LLC, United States; ORCID 0009-0000-9877-3623. Correspondence to: Andrew Hendel <ajhendel@latentarchon.com>.

1.1 Inference: any model, the same bits on every vendor

A model does not need to be trained in Mojolearn to be served by it. Transformer and Mamba blocks load weights trained in other frameworks, and the same weights and inputs then return the same output bits on Apple, NVIDIA and AMD GPUs and on CPUs. A team can test a deployed model on different hardware, replay a recorded decision during an audit, and replace serving hardware without numerical change. The guarantee covers the model once it runs in Mojolearn. The framework that trained it rounds differently, and its own outputs vary from GPU to GPU, so Mojolearn’s outputs can differ from that framework’s in the last bits.

1.2 Training: the same model on every vendor

Training in Mojolearn is also bitwise identical across vendors. Given the same code revision, data, hyperparameters and seed, Apple, NVIDIA and AMD GPUs produce the same model state, gradients, optimizer state and checkpoint bytes. A checkpoint written on one vendor can therefore resume on another and follow the same trajectory as uninterrupted training. An identical model gives identical answers only while it is also served in *identical* mode: if the trained weights are served by another framework, that framework’s arithmetic applies and outputs can again differ from GPU to

GPU.

1.3 Portability is inherited; identity is not

Mojo and MAX provide compilation and execution across Metal, CUDA and HIP (Modular, 2026a; Lattner et al., 2021); they do not make the arithmetic agree across those backends. Mojolearn defines the shared numerical rules and implements the algorithms that obey them. Its own numerical kernels control arithmetic and reduction order, replacing vendor math routines where their results would depend on the backend. The contribution is the numerical contract and its implementation across the library’s algorithms; Mojo and MAX supply the execution substrate. Section 5 explains how the contract is enforced.

Bitwise identity matters throughout the library, beyond neural networks. In a tree learner, rounding can change the winning split and every subsequent node; in k -means, it can move a point to another cluster and change every later centroid. These are applications of one numerical contract across many algorithms.

1.4 Contributions

- (a) *Bitwise-identical machine-learning algorithms*: a single Mojo codebase supports training and inference for over 100 algorithms in 13 families, from decision trees to neural networks.
- (b) *Cross-vendor handoff and mixed-vendor training*: a training run can move from one GPU vendor to another mid-run, or be shared by Apple, NVIDIA and AMD GPUs working on one model at once. Either way, the training state after every step is bitwise identical to a single GPU’s (Sections 7.1 and 7.5).
- (c) *A numerical contract that separates arithmetic from scheduling*: the contract fixes the operations that determine model bits, including reduction order, rounding and elementary functions. Tiling, thread layout, memory placement and kernel scheduling remain free, so each backend can be tuned for its own hardware without changing a bit (Section 5). This vendor-specific tuning has only begun.
- (d) *Apple M-series GPU support*: every algorithm runs on the GPUs of Apple M-series Macs, where XGBoost, LightGBM, CatBoost and cuML have no supported GPU path and scikit-learn reaches the GPU for only a few estimators (Section 8).

To my knowledge, these are the first cross-vendor bitwise-identical GPU training and inference implementations. I achieve this in full FP32 floating-point arithmetic for neural networks, without reducing the computation to integers. Prior bitwise results are single-vendor or emulate one vendor’s arithmetic (Section 8).

Building a cross-vendor numerical profile required an extensive implementation effort. Mojo 1.0 was released on August 11, 2026; development of Mojolearn began on August 19, 2026, and the original preprint was submitted to arXiv on September 8, 2026. Release 0.7.0, archived on September 9, 2026 (Hendel, 2026c), documents three-vendor training identity for a byte-level language model. As of October 8, 2026, Mojolearn comprises 758,203 lines of original Mojo code: 446,531 lines of library implementation and 311,672 lines of verification fixtures, test harnesses and benchmarks, with a further 387,930 lines of Python, shell and other code (Appendix C). Its public repository holds 14,912 commits as of October 8, 2026. The system was built with AI coding assistants under the author’s direction, and every result reported here is checked by the verifier, not taken from the code’s authorship. This scale illustrates the implementation effort required to cover the library’s algorithm families.

2 THE SYSTEM

Users call Mojolearn from Python through scikit-learn-shaped estimators (Pedregosa et al., 2011) and neural blocks. One Mojo source compiles for Metal, CUDA and HIP and ships as an Apple silicon wheel and a Linux x86-64 wheel. In identical mode, all numerical work, including the portable elementary functions, runs in the library’s own compiled Mojo kernels (Modular, 2026a), without calling MAX’s matrix-multiplication kernels, vendor BLAS and solver libraries, or external ML frameworks.

2.1 The user-facing contract

One flag, `numeric_mode="identical"`, selects the default numerical contract on every supported interface. For example, a caller passes weights to a transformer block and runs its forward pass in identical mode. A fresh process using the same input and weight bytes, hyperparameters, seed and numerical profile returns the same bits on a verified device.

2.2 Algorithms and interfaces

Table 1 summarizes the neural, tree and classical interfaces, including random forests and extremely randomized trees (Breiman, 2001; Geurts et al., 2006). Transformer and Mamba blocks support forward computation and stateful decoding; fixed-shape training interfaces support a small MLP and a byte-level language model (Section 7.1). Every interface also runs on x86-64 and Arm CPUs, with the CPU reference build included in library-wide verification. Neural blocks expand BF16 and INT8 weight storage to FP32; the separate INT8 arithmetic profile uses integer accumulation (Section 5).

The breadth tests the contract’s generality: each family exposes distinct ways for hardware-dependent arithmetic to alter models or outputs (Table 13).

The implementations build on published designs:

family	algorithms and components
boosting	symmetric-tree, depth-wise, loss-guided and ordered boosting; AdaBoost, DART
trees and ensembles	decision trees, random forests, extremely randomized trees, isolation forests, bagging, voting, stacking
clustering	k -means, mini-batch and bisecting k -means, DBSCAN, HDBSCAN, agglomerative, spectral, mean shift, OPTICS, affinity propagation, Gaussian and Bayesian Gaussian mixtures
neighbors, density and search	nearest neighbors, k -NN, radius neighbors, random ball cover, kernel density, local outlier factor, label propagation, IVF-Flat, IVF-PQ, CAGRA
linear models	least squares, ridge, lasso, elastic net, logistic regression, SGD, Huber, Poisson, Gamma, Tweedie, Bayesian ridge, ARD, Lars, quantile and isotonic regression
kernel methods and Gaussian processes	SVMs, kernel ridge, Gaussian process regression and classification, sparse variational GP, Nyström, random Fourier and chi-square features
decomposition and manifold	PCA, incremental and kernel PCA, truncated SVD, NMF, ICA, factor analysis, PLS, CCA, dictionary learning, sparse PCA, latent Dirichlet allocation, UMAP, t -SNE, Isomap, MDS, LLE
time series	ARIMA, AutoARIMA, Holt-Winters, ETS, STL, VAR, Theta, Croston, GARCH, Prophet-style forecasting, KPSS
preprocessing, probabilistic models and resampling	scalers, encoders, imputers, discretization, feature selection, naive Bayes, discriminant analysis, bootstrap, permutation tests, Monte Carlo integration
neural blocks and optimizers	transformers, Mamba-1/2/3, Samba, MLPs, RNNs, LSTMs, GRUs, mixture of experts, Adam, AdamW, SGD, RMSprop, Adagrad, Lion, Adafactor, LAMB
convolutional and graph networks	1-D and 2-D convolutions, pooling, batch normalization, residual blocks, CNN classifiers, GCN and GraphSAGE convolutions
language models and tokenization	decoder language models, byte-level BPE tokenization
linear algebra	matrix products, Cholesky, QR, LU, eigendecomposition, SVD, least squares
also provided, not counted	evaluation metrics; cross-validation; CPU inference for saved models; stepwise decoding with explicit state; checkpoint-loaded causal language models; BF16 and INT8 weight storage; tokenized corpora; multi-GPU execution plans; the verifier

Table 1. Over 100 algorithms across 13 families. The table lists representative methods and model components from the current public API, grouping related estimator variants. A classifier and a regressor of the same method count once; losses, kernels, distance metrics, penalties, starting rules and multi-GPU execution are configurations, not separate algorithms. The fixed verification record and its neural configurations are reported separately in Section 7 and Table 10.

CatBoost’s oblivious, depth-wise and loss-guided trees (Prokhorenkova et al., 2018); fixed-point gradient histograms and feature sampling from the XGBoost and LightGBM GPU learners (Chen & Guestrin, 2016; Ke et al., 2017); and cuML, RAFT and cuVS designs for forests and classical methods (RAPIDS Development Team, 2026). The contribution is the numerical contract they share and the kernels that enforce it, not new learning algorithms.

2.3 Numeric modes

Identical is the default. Classical, tree and neural estimators also offer an opt-in *fast* mode, which gives up bit agreement for speed; it is tuned for Apple GPUs, so that classical machine learning and tree models can make full use of a laptop’s GPU. Gradient boosting, random forests and extremely randomized trees also offer *deterministic* mode, which repeats bits on one device but not across vendors. Table 6 summarizes the three modes, and Figure 1 shows one computation that repeats on each device yet differs across vendors.

3 SCOPE AND LIMITATIONS

The guarantees below require the same code revision, numerical profile, input bytes, initial state, hyperparameters and seed, with identical mode selected and the configuration covered by the verification record.

3.1 Scope of inference

Every verified device returns the same output bits for the supplied model weights, whether those weights were trained in Mojolearn or elsewhere. Inference is also batch invariant (He & Thinking Machines Lab, 2025): a sequence’s output bits do not depend on how many other sequences share the batch.

3.2 Scope of training

Every verified device produces the same learned model and supported training state, including gradients, optimizer state and checkpoint bytes. A refused configuration never counts as a pass.

3.3 Limitations

These limits apply throughout; the rest of the paper does not repeat them.

- *Devices.* The record was collected on an Apple M4, on NVIDIA H100, A100 and RTX 4090 GPUs, on AMD MI325X and MI300X GPUs, and on AMD EPYC and Apple M4 CPUs; Appendix A lists the further devices used outside the record. Mojo supports GPU programming on Apple M-series GPUs, substantially all NVIDIA GPUs via PTX and select AMD GPUs (Modular, 2026b). A GPU is not covered until it passes the same checks (Table 2).
- *Precision.* The guarantee is for FP32 arithmetic. FP64 input is converted or refused.
- *What was tested.* The record covers the configurations, inputs, model shapes and release it names. A call that succeeds on an untested configuration is not a verified result.
- *Bitwise-identical inference.* Inference must run in identical mode. A result from fast mode or another framework is not covered.
- *Option coverage.* Some interfaces accept more options than the record covers (Appendix A.4).
- *Input boundary.* The guarantee begins at the estimator input; preprocessing performed outside the library is not covered.
- *Quality.* Identical bytes can encode a poor model, so quality validation remains a separate obligation.
- *Speed and cost.* Performance is evaluated in Section 5.8. Mojolearn has been in development only since August 2026 and has not received the years of tuning that established libraries have, so its current speed reflects the maturity of the implementation as well as the cost of identity.

Table 2 also lists prospective Qualcomm Adreno and Intel Xe hardware against the contract’s requirements. Each new backend must pass verification; hardware that violates a pinned assumption is refused. Changes to the numerical rules require a new profile version.

4 WHY GPUS PRODUCE DIFFERENT MODELS

GPU backends can differ in intermediate precision, reduction grouping and elementary functions even with identical models, inputs and seeds. Neural networks propagate these differences through outputs and training state; tree learners can turn them into structural changes.

4.1 Neural inference

Transformer matrix products depend on product precision and summation order; attention and normalization add division, exponentials and square roots. Every operation must obey shared numerical rules. An activation difference can reach the logits, change the ordering of nearly tied tokens, and alter even greedy decoding and its subsequent inputs.

Mamba blocks also carry recurrent state between decoding calls. Matching weights and current inputs cannot restore agreement if that state already differs, so identity must cover both outputs and retained state.

4.2 Neural training

Backward matrix products, gradient reductions, clipping and optimizer updates can change parameters and AdamW moments that feed the next step. A common numerical profile must therefore cover forward computation, loss, backward computation and updates. Checkpoint continuation also needs the same parameters, moments, counters and subsequent inputs. These dependencies motivate comparing intermediate outputs and training state alongside final checkpoint bytes (Section 7.1). Even a difference that leaves model quality unchanged breaks exact replay.

4.3 Tree training

CatBoost’s GPU histogram implementation uses floating-point atomics, so thread arrival determines summation order. Its partition-statistics kernels derive chunk counts from the number of streaming multiprocessors, making device geometry a summation choice (CatBoost Developers, 2026a). Either can change split scores and, near a tie, select another partition and alter subsequent trees. Border selection, leaf updates and tie-breaking must obey the same numerical rules as histogram accumulation.

Table 13 identifies the corresponding hazards in the other algorithm families. Section 5 explains how the shared numerical contract controls them.

5 HOW IDENTICAL MODE WORKS

5.1 Numerical rules and hardware differences

Identical mode controls every operation that can change model bits in one of three ways. It *pins* settings the hardware would otherwise choose, such as the number of blocks a sum is split into; it *replaces* functions that GPUs compute differently, such as square root, with its own portable implementation; and it *refuses*, with an explanatory error, a configuration it cannot make identical, such as an unsupported dtype, shape or option. Operations that do not change arithmetic order or precision remain free to use each backend’s normal scheduling. Appendix B.3 lists every place these rules apply (Table 14).

	verified, measured			buildable, unvalidated	prospective, not buildable	
	Apple M4	NVIDIA H100	AMD MI325X (CDNA)	AMD RDNA	Qualcomm Adreno	Intel Xe
square root	correctly rounded	approximate	correctly rounded	—	—	—
denormals	flushed	kept	kept	—	—	—
$a \cdot b + c$	fused	fused	fused	—	—	—
$\max(+0, -0)$	-0.0	+0.0	+0.0	—	—	—
NaN payload	0x7fc00000	0x7fffffff	0xffc00000	—	—	—
lockstep lane width	32	32	64	32	8, compiler-chosen	8, compiler-chosen
vendor FP32 matrix product	FP32	TF32 by default	FP32	FP32	FP32	FP32
float atomics	global only	yes	yes	yes	no	yes
threadgroup memory per block	32 KB	48 KB	64 KB	64 KB	32 KB	64 KB
meets minimum hardware requirements	yes	yes	yes	yes	yes	yes

Table 2. Current GPU coverage and requirements for future backends. Arithmetic results are measured on Apple M4, NVIDIA H100 and AMD MI325X. AMD RDNA is buildable but unvalidated; Adreno and Xe entries describe prospective targets from vendor documentation, unsupported by the evaluated toolchain. The minimum hardware requirements are 32 KB of threadgroup memory, threadgroup integer atomics and 1024-thread blocks. Meeting these requirements is only a starting point: each new backend must pass the same verification. Identical mode supplies shared arithmetic where hardware behavior differs, including square root, denormal handling, matrix products and a logical reduction width of 32 lanes.

5.2 Machine parameters can change the result

A block count, a replication factor and a reduction width all determine which partial sums are combined. They are therefore numerical choices, even when an implementation labels them as scheduling or tuning parameters. This is established in the reproducible-summation literature (Demmel & Nguyen, 2013; Demmel et al., 2016; Collange et al., 2015), appears as a product constraint in vendor math libraries (Lubin & Mueller-Albrecht, 2024), and is restated for inference as batch invariance (He & Thinking Machines Lab, 2025). Identical mode pins every machine-derived parameter that reaches floating-point arithmetic.

5.3 Floating-point and integer matrix units differ

The FP32 profile avoids vendor floating-point matrix units: their fragment shapes and internal accumulation do not implement the library’s specified rounding order, and TF32 additionally rounds operands before multiplication. RepDL identifies hardware-specific low-precision matrix arithmetic as a similar obstacle (Xie et al., 2025). My measured FP32 workloads therefore retain the cost of the general-purpose arithmetic path. The separate INT8 profile admits integer matrix units because its bounded Int32 products and sums are exact before the pinned conversion to floating point. Changing the grouping of an overflow-free integer sum cannot change its value. The FP32 workloads do not use those units.

5.4 Ordinary-looking choices change bits

DBSCAN normally sizes its batches from free device memory, so unrelated memory use could change the graph decomposition; identical mode pins the batch size. A GPU’s default square root may be an approximate instruction, so identical mode uses its own implementation on every backend. Table 2 summarizes the measured arithmetic differences that portable replacements must cover.

5.5 Neural matrix products

The FP32 matrix-product profile used by transformer projections separates the logical arithmetic from its physical execution. Each output element is a reduction over K products. The profile partitions that reduction into contiguous leaves whose length depends on K alone: K itself up to 128, then 128 until the leaf count would exceed 1,024, then $\lceil K/1024 \rceil$. Neither batch size nor GPU occupancy can change these constants. Within a leaf, the products are accumulated in ascending order from positive zero by a fused multiply-add that rounds once to FP32, and the accumulator is flushed to zero of the same sign after every step. Flushing only the final partial would allow a backend that retains subnormals to carry a different intermediate into the next update. The leaf partials are then combined in a fixed binary tree over adjacent pairs; an unpaired final partial is carried unchanged to the next level rather than padded with zero, and an empty reduction returns positive zero. GPU threads may compute leaves or output tiles concurrently, and intermediate values may live in registers, shared memory or a separate buffer, but each output must follow this same dependency tree. A GPU whose hardware runs 64 threads in lockstep, where the others run 32, cannot substitute its native reduction for the prescribed pairing. Thus changing tile size or batching changes the schedule without changing the arithmetic; changing a leaf boundary, pairing or rounding step requires a new numerical profile. The repository’s contract document for this profile states each rule together with the fixture that checks it.

5.6 Tree histograms and splits

The fixed-point histogram path must first turn each floating-point statistic into the same integer on every device. It scales the gradients and Hessians by a power of two chosen from the row count and from fixed-order sums of their magnitudes, so that every subset sum, with an allowance of one

quantization unit per row, fits in a signed Int32. Bounding magnitudes rather than the signed total also bounds intermediate sums when positive and negative gradients cancel. Each scaled statistic is quantized with deterministic dither: a hash of the row key, independent of thread arrival order, decides whether the fractional part rounds up, which reduces the systematic bias that truncating equal statistics would introduce. Integer atomics then accumulate the bounded values in any arrival order, because an overflow-free integer sum is exact, and the total is converted back to floating point once, at the histogram boundary; converting partials early and summing them as floats would forfeit that order independence.

Identical histograms are necessary but insufficient for identical trees. Partition totals and split-score expressions retain their pinned folds and rounding rules, and the final comparison orders candidates by gain, then feature index, then bin index, in a fixed traversal order; a full tie keeps the earlier candidate. The discrete split that determines the next partition is therefore the same on every device. Fixed-point quantization changes the numerical computation; its resolution and the resulting model quality remain separate from the identity guarantee. Together, the examples illustrate the scheduling rule: prescribe the order where rounding matters, and permit order changes where bounded integer arithmetic is exact. Appendix B.3 places these mechanisms in the library-wide numerical contract.

5.7 Training handoff

The supported language-model trainer checkpoints parameters, optimizer moments, initialization flags, the completed-step counter and optimizer settings as raw state. Companion metadata binds that checkpoint to the numerical profile, code revision, corpus and token schedule, including the position at which training resumes. The receiving device restores the same state and consumes the same next batch with the same logical microbatch partition. Forward computation, loss, backward computation and the optimizer update follow the shared numerical rules, so the next state matches the uninterrupted run. Repeating this argument gives the same subsequent trajectory for supported computations. Transferring weights alone would not suffice: different moments or a different step counter can change the next update even when the forward outputs agree.

5.8 What Identity Costs

Cross-vendor bitwise identity carries implementation and execution costs: identical mode gives up vendor matrix units, approximate elementary functions and device-chosen reduction geometry. These constraints require additional implementation work and can increase execution time. Table 3 compares representative identical-mode fits with the fastest GPU opponent that completed on the same machine;

Appendix D lists every cell with its time and quality. The representative neural and dense linear-algebra workloads are slower than their listed GPU opponents; the fastest completed neural baselines use reduced-precision BF16 or TF32 matrix units. These ratios compare whole implementations, each with its own algorithm, defaults and clock; they do not mean that identity is free.

6 REPRODUCIBILITY AND VERIFICATION

6.1 Verification checks

The installed verifier provides three complementary checks:

- *Against recorded references.* The verifier trains each covered algorithm on fixed fixtures and compares training-state, held-out-prediction and saved-model hashes with Apple, NVIDIA, AMD and CPU references. Matching input hashes are required before numerical comparison; an agreeing machine joins the recorded columns.
- *GPU against CPU on the user’s own machine.* Each model is fitted on the user’s GPU, then held-out predictions are compared with those from the saved model reloaded on the CPU. This checks inference without trusting a recorded reference and runs on any supported GPU, including a laptop’s. It does not independently repeat training on the CPU.
- *Two independent parties.* Two people run the verifier on their own machines and compare evidence documents without a GPU or the author’s involvement. Each can publish a commitment before exchanging documents to prevent copying after seeing the other result. The comparison rejects mismatches, self-contradictions, uncomputed cells and a document compared with itself.

The checks share the library’s implementation and establish cross-device consistency; algorithmic correctness and model quality require independent references and quality criteria. Verdicts distinguish agreement, divergence, missing references, refusal and non-applicability; only agreement counts as verified. A self-test accepts an unchanged fixture and detects an input differing in its last bit.

Where applicable, saved models must reproduce in-memory predictions. Rows and per-row gradients must remain batch invariant, and padded or ragged sequences must match isolated execution. Token-by-token decoding must equal a full forward pass; gradient accumulation must agree for splits preserving the reduction tree; and sampler log-probabilities must match trainer recomputation. Transformer and Mamba checks span batch sizes 1–1,024 and sequence lengths on either side of every kernel’s chunk boundary. Numerical parts unsupported by an interface are explicitly non-applicable. Training state is compared at every recorded stage because

family	workload	NVIDIA L40S		AMD MI325X	
		fastest GPU opponent	ours/theirs	fastest GPU opponent	ours/theirs
Neural	language-model training step	torch-compile-bf16	4.32	torch-compile-bf16	15.10
	transformer forward pass	torch-eager-bf16	1.90	torch-eager-bf16	9.35
	Mamba-2 forward pass	mamba-ssm-tf32	4.58	torch-eager-bf16	2.03
Trees	symmetric (oblivious) boosting	catboost-gpu	0.75	–	–
	depthwise boosting	xgboost-gpu	0.77	–	–
	random forest	cuml-rf-gpu	1.28	–	–
Classical	k -means	cuml-gpu [†]	1.66	torch-gpu [†]	0.39
	PCA	torch-gpu [†]	13.73	torch-gpu [†]	30.96
	ridge regression	cuml-gpu [†]	16.58	–	–

Table 3. Representative identical-mode fit times against the fastest GPU opponent that completed on the same board (ratio of medians; below 1 means ours took less time; “–” where no GPU opponent completed on that board). Datasets: neural rows bytes, gaussian; trees rows istella; classical rows istella. [†] the opponent’s clock starts with its inputs already on the device and excludes the host-to-device copy that our clock includes. [‡] the opponent’s model was fitted before its clock started. [§] the harness marked this comparison not comparable (the two models differ materially in size, for example in leaf count), so the ratio is not a like-for-like time. Tree rows on the L40S come from the main board of current code (updated 2026-10-09), with opponents from the stored opponent boards. Every cell, with times, quality and the Apple boards, is in Appendix D.

#	stage	type	n	Apple M4	NVIDIA H100	AMD MI325X
0	knn.index_norm	f32	20000	74a472e4925ec44e	74a472e4925ec44e	74a472e4925ec44e
1	knn.query_norm	f32	2000	18da124518cd0e52	18da124518cd0e52	18da124518cd0e52
2	knn.out_dist	f32	20000	18499d126dfd6ffe	563af8012606fe61	18499d126dfd6ffe ◀
3	knn.out_idx	u32	20000	379df04034c318b9	379df04034c318b9	379df04034c318b9
4	knn.sorted_dist	f32	20000	18499d126dfd6ffe	563af8012606fe61	18499d126dfd6ffe ◀
5	knn.sorted_idx	u32	20000	379df04034c318b9	379df04034c318b9	379df04034c318b9

Figure 1. Optional trace hashing identifies the first recorded divergence in an unpinned k-NN computation. The standard-library path first differs on NVIDIA at the distance buffer (◀), and that difference survives in its sorted copy; the preceding norms and resulting neighbor indices agree. This is CUDA’s approximate PTX square root. Identical mode uses Mojolearn’s pinned square root and produces matching bits without requiring trace hashing during normal training.

different models can agree on predictions. Table 5 counts coverage by property. The release 0.8.10 coverage inventory maps its 273 verifier configurations to these checks; documents retain binary digests and provenance (Appendix A). Current API breadth is separate (Table 1).

6.2 Optional trace hashing

Verification and debugging builds can record a *trace card*: hashes of the raw buffer bits at each named stage, so that comparing two cards locates the first recorded stage that differs. Figure 1 uses one to isolate CUDA’s approximate square root in an unpinned k-NN computation; normal training omits this instrumentation.

7 RESULTS: THE SAME MODEL ACROSS THREE VENDORS

The verification record for release 0.8.10 combines independently recorded Apple Metal, NVIDIA CUDA, AMD HIP and CPU results. Across 214 single-device configurations and nine fixtures, all 7,452 applicable numerical parts agree

bit for bit across the CPU and all three GPU vendors, with no mismatches or missing numerical vendor comparisons. Another 18 saved-model parts agree across all three GPUs; the CPU loads those models rather than writing them. The fixed evidence archive contains the reference table, its underlying records and the machine-readable coverage audit (Appendix A). Separate identical-mode benchmark boards for release 0.8.25 report six NVIDIA–AMD output-hash mismatches (Appendix D).

A *configuration* is one algorithm with one set of options. A *cell* is one configuration on one fixture; its *parts* compare training state, saved-model bytes, held-out predictions and applicable batch or sequence properties. The single-device record contains 1,926 cells. Appendix A gives their coverage by family (Table 4) and the counts by part and fixture. The reference table also contains 59 two-GPU configurations, reported separately in Section 7.5; they are not included in these totals.

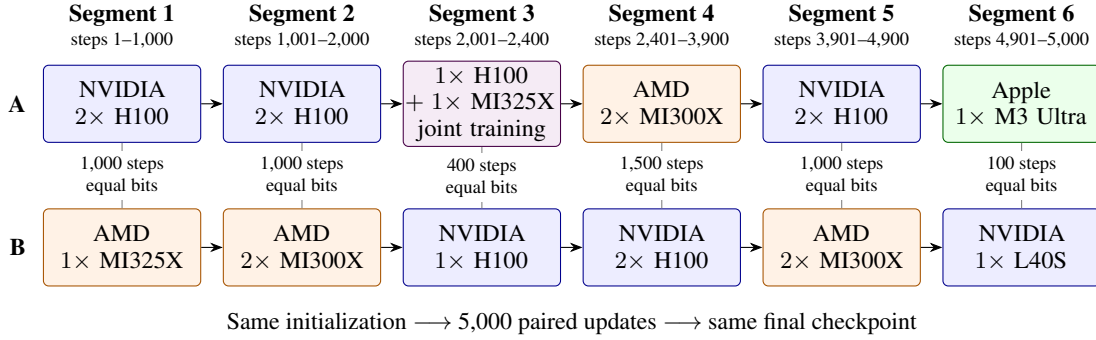


Figure 2. Two hardware routes through the same GPT-3 Small-shaped training workload: 162,147,840 parameters and 2,621,440,000 tokens per route. Boxes name the GPU model and number of participating devices; arrows denote checkpoint continuation, and vertical comparisons cover every step in each segment. A/3 trains jointly on an H100 and an MI325X in separate machines. All H100s are the 80 GB HBM3 model. The two routes agree on state and summed gradient digests, all 64 per-shard loss bit patterns, and learning-rate bits at all 5,000 steps; their final checkpoint file digests also match. Segment widths are schematic, not proportional to steps or elapsed time.

7.1 Bitwise-identical neural inference and training

The neural configurations apply the same contract to forward computation and to training. Transformer, Mamba-1, Mamba-2, Mamba-3 and hybrid blocks match across Apple, NVIDIA and AMD in outputs, recurrent state and gradients. Training configurations run embedding, forward and backward passes, cross-entropy loss and AdamW with gradient clipping, and the three vendors write byte-identical checkpoint files without consuming another run’s rounding log. The contract therefore composes through gradient computation, optimizer updates and checkpoint serialization.

The record includes full and windowed transformers, the three Mamba variants, hybrid stacks, MLPs, language-model training and reduced-precision weight storage. Table 10 reports configurations by group; Table 5 separates forward, saved-model, gradient and sequence properties so that a match on one property is not presented as coverage of another.

A separate byte-level language-model demonstration trains the same model independently, from initialization to the final step, on each of the three vendors, and compares every recorded array of every training step. No run replays another vendor’s recorded steps. A checkpoint resumes on another vendor and rejoins the uninterrupted trajectory in both directions, and CPU replay reproduces individual recorded training steps. Appendix A.5 describes this demonstration separately from the fixture record.

7.2 A GPT-3 Small-shaped model across six segments

A 162,147,840-parameter decoder trains for 5,000 optimizer steps on 2,621,440,000 FineWeb-Edu tokens along each route in Figure 2. Route A uses NVIDIA, AMD and Apple GPUs; route B uses AMD and NVIDIA GPUs, with a different hardware configuration at every corresponding segment. Both routes start from the same initialization and

token schedule; each then continues from its own complete checkpoints. Route B computes its own updates and compares them with route A’s recorded state and gradient digests, per-shard loss bits and learning-rate bits at every optimizer step. Those reference records supply comparison values, not gradients or rounding corrections. At handoffs, an arrival replay recomputes the preceding two steps before training continues.

All 5,000 paired steps agree in the recorded state, summed gradient, 64 per-shard losses and learning rate, with zero divergent steps, so the trajectories agree and not only the final checkpoints. The two workers in A/3 also agree at all 400 jointly computed steps. The six segment endpoints and all 56 common saved checkpoint digests agree across routes, including the final checkpoint written by the Apple M3 Ultra and NVIDIA L40S. This scale experiment is separate from the release 0.8.10 fixture totals; its step records, handoff evidence and negative controls appear in Appendix A.5.

7.3 Trees, classical models and the full library

The same contract holds beyond neural networks, for gradient boosting, forests, k -means, DBSCAN, PCA, linear and kernel models, ARIMA and the other classical families in Table 4 (Appendix A). Each configuration is fitted on nine fixtures, a base case and adversarial cases built from many-way ties, unstructured hashed values, column magnitudes spanning eight orders, subnormal values, duplicated and constant columns, sizes that no block divides and all-negative inputs (Table 7). Models trained on one vendor load and produce bitwise-identical predictions on the others. A configuration the contract cannot cover is refused by name instead of being approximated, and such a refusal is recorded as a refusal, never as a match.

7.4 The verifier can fail

A table of matching hashes is only evidence if a wrong implementation would have produced a mismatch. Each family therefore has negative controls, builds of the library with one deliberate arithmetic change such as a reversed fold, a transposed operand or a dropped flush to zero. A control counts only when a committed pair of runs shows that it moved the hash of that configuration. A control that exists in the source but has not been seen to move a configuration counts for nothing. By that rule, a changed build has been observed to move the bits of 205 of the 214 single-device configurations; the other nine have a declared control that has not yet been observed to move them (Appendix A.3). The verifier’s own self-test also detects an input that differs only in its last bit.

7.5 Two-GPU execution

Changing the device count can change the answer when it changes how sums are grouped. Mojolearn instead divides work into pieces that each GPU computes exactly as a single GPU would, such as whole trees in a forest or whole microbatches in neural training, and combines them in a fixed order. The reference table contains 59 two-GPU configurations covering classical and neural workloads. All 1,935 of their numerical parts have equal bits across the Apple, NVIDIA and AMD columns, with no missing value and no disagreement, and the 771 parts that have a numerical CPU role match the CPU as well (Appendix A.4).

Two-GPU execution is also checked on two physical GPUs. The shipped verifier runs each configuration once on one GPU and once across two, confirms by device identifier that two distinct GPUs were used, and fails if a single bit moves. With release 0.8.10 installed from PyPI on two NVIDIA RTX 4090 GPUs, the verifier’s self-test passes, its quick check of one configuration per family is verified, and a sweep of all nine fixtures compared 1,935 parts between one and two GPUs: all are identical and none diverges. On two AMD MI300X GPUs, two-GPU records equal the same reference values. More than two devices in one machine and multi-GPU throughput are not measured. The same fixed fold also lets an Apple, an NVIDIA and an AMD GPU in separate machines train one model together with identical state after every step (Appendix A.5). An Apple M4, an NVIDIA H100 and an AMD MI300X trained one small decoder for 6 steps: the three workers reported the same state hash after every step, equal to a single GPU’s, and all 1,512 assignments of the recorded microbatch gradients to vendors reproduce the same sum.

8 RELATED WORK

8.1 Reproducible arithmetic

Device geometry and partitioning determine summation order, as established (Demmel & Nguyen, 2013; Demmel et al., 2016; Lubin & Mueller-Albrecht, 2024; He & Thinking Machines Lab, 2025), and integer gradient histograms predate this work (Shi et al., 2022). ReproBLAS and Intel MKL’s conditional reproducibility mode provide reproducible summation and linear algebra (Demmel et al., 2016; Intel, 2024). Climate models demonstrate identity across compiler versions and flags (Li et al., 2016); molecular-dynamics codes document processor-count sensitivity (OpenMD, 2013). My arithmetic probes confirm documented differences in denormals, FMA contraction and elementary functions (Dawson, 2013; Whitehead & Fit-Florea, 2011).

8.2 Deterministic modes

Framework deterministic modes switch known-nondeterministic operations to deterministic ones or refuse them (PyTorch, 2026; TensorFlow, 2024). NVIDIA’s primitive library grades reductions as nondeterministic, run-to-run deterministic or GPU-to-GPU deterministic (Al Awar & Singanaboina, 2026). Environment capture and replay also support repeatable training (Chen et al., 2022; Heumos et al., 2023). Within-platform repeatability does not establish cross-vendor identity (Figure 1).

8.3 Bitwise training

RepDL (Xie et al., 2025) pursues cross-system reproducibility with correctly rounded FP32 operations, fixed summation order and deliberate FMA use. It provides CPU and CUDA implementations and a reproducible training example; the report has no evaluation section and names no tested hardware models. Optimistic Verifiable Training (Srivastava et al., 2024) reproduces training across three NVIDIA GPUs by recording rounding decisions for an auditor to replay. Verde’s RepOps (Arun et al., 2025) fixes CUDA operation order across the NVIDIA configurations it tests; Hawkeye (Badash et al., 2026) emulates NVIDIA Tensor Core arithmetic on CPUs.

Concurrently, OPEN-1B (Donaghy et al., 2026) trains a 1.6-billion-parameter model on 48 NVIDIA H100 GPUs. Auditors replay individual steps bit for bit on a CPU, NVIDIA GPU or Apple GPU against published per-step hashes, with many auditors intended to cover the run collectively. Its large matrix products use INT8 with Int32 accumulation; it also controls floating-point reductions, fusion, subnormals and optimizer arithmetic. It covers one architecture and no AMD GPUs, with training on NVIDIA and replay on other devices. Mojolearn keeps neural matrix products in FP32 and compares complete independent training runs on each vendor.

8.4 Bitwise inference

EigenAI (Alves et al., 2026) fixes GPU architecture and reports different A100 and H100 outputs; Cankaya (2026) emulates NVIDIA inference on CPUs. Batch- and parallelism-invariant kernels support deterministic serving (He & Thinking Machines Lab, 2025; Zhang et al., 2025; vLLM Project, 2026); LLM-42 instead uses existing kernels with scheduling and verify-and-rollback (Gond et al., 2026). Gensyn’s REE packages reproducible operators and checkable receipts on NVIDIA GPUs and CPUs (Gensyn, 2026; 2025). Power-of-two INT8 scales make two quantized kernels agree on one GPU (Chen, 2026a;b). One Digest’s self-published report (Charlot, 2026) covers bit-identical FP32 transformer forward kernels on Apple Metal, an NVIDIA GPU through Vulkan, WebGPU and three CPU targets, with no training, AMD or classical learning. HEAL (Zhu et al., 2026) reduces numerical divergence and answer flips on NVIDIA and AMD without claiming bitwise identity.

8.5 Portable GPU programming

Kokkos and Triton provide portable GPU programming (Edwards et al., 2014; Tillet et al., 2019); Mojo and MAX supply this work’s substrate (Lattner et al., 2021; Modular, 2026a; Johnson et al., 2024), with performance portability shown on H100 and MI300A (Godoy et al., 2025). Meganeura (Malyszau, 2026) trains through Vulkan and Metal on NVIDIA, AMD, Apple and Intel GPUs, validating against error thresholds. Mojmemo (yetalit, 2024) implements classical ML in Mojo for CPUs, with no GPU backend or identity contract. GPU boosting libraries and research (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018; Wen et al., 2020; 2019; Zhang et al., 2017) and RAPIDS (Raschka et al., 2020) lack Apple-silicon GPU support (XGBoost Developers, 2026; LightGBM Developers, 2026; CatBoost Developers, 2026b; NVIDIA, 2026). MacBoost (Rowley, 2026) trains boosted trees through Apple Metal kernels, targeting single-vendor speed without a training-identity claim.

9 CONCLUSION

The same machine-learning workload can produce different bits on different GPUs. Mojolearn removes that variable. In its default identical mode, one numerical contract, implemented once in Mojo, gives the same model and output bits across verified Apple, NVIDIA and AMD GPUs and x86-64 and Arm CPUs, for over 100 machine-learning algorithms in 13 families, from decision trees to neural networks, in both inference and training. A model trained elsewhere returns the same answers on every verified device, a training run can move between GPU vendors, or be shared among them, without leaving its trajectory, and anyone can check these claims on their own hardware, including against their own CPU. Identity has implementation and execution costs.

Future work. The next steps are making identical mode faster, extending algorithm and option coverage, verifying more GPUs, and stabilizing the library for a 1.0 release. Bounded integer arithmetic offers further optimization opportunities under the conditions described in Section 5.

REFERENCES

- Al Awar, N. and Singanaboina, S. Y. Controlling floating-point determinism in NVIDIA CCCL. NVIDIA Technical Blog, March 2026. URL <https://developer.nvidia.com/blog/controlling-floating-point-determinism-in-nvidia-cccl>.
- Alves, D. R., Patankar, V., Pereira, M., Stephens, J., Vaziri, N., and Kannan, S. EigenAI: Deterministic inference, verifiable results, 2026. URL <https://arxiv.org/abs/2602.00182>. arXiv:2602.00182 [cs.CR].
- Arun, A., St. Arnaud, A., Titov, A., Wilcox, B., Kolobaric, V., Brinkmann, M., Ersoy, O., Fielding, B., and Bonneau, J. Verde: Verification via refereed delegation for machine learning programs, 2025. URL <https://arxiv.org/abs/2502.19405>. arXiv:2502.19405 [cs.LG].
- Badash, E., Boneh, D., Komargodski, I., and Srivastava, M. Hawkeye: Reproducing GPU-level non-determinism. In *MLSys*, 2026. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/e217c271a57c365a246b0ad39e668ba8-Abstract-Conference.html.
- Breiman, L. Random forests. *Machine Learning*, 45(1): 5–32, 2001. doi: 10.1023/A:1010933404324.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020.
- Cankaya, N. Bit-exact AI inference verification without performance tradeoffs, 2026. URL <https://arxiv.org/abs/2606.00279>. arXiv:2606.00279 [cs.CR].
- CatBoost Developers. CatBoost gpu histogram and partition-statistics kernels, 2026a. URL <https://github.com/catboost/catboost/tree/7055d33dd10bc5960bc5a32e4d884dab1a1356e8/catboost/cuda>. Source code, revision 7055d33.

- CatBoost Developers. pip install. CatBoost documentation, Python package installation, 2026b. URL <https://catboost.ai/docs/en/installation/python-installation-method-pip-install>. Supported-platforms table lists no GPU support on macOS; accessed September 20, 2026.
- Charlot, D. J. One digest: Bit-identical neural computation across heterogeneous compute fabrics. Technical Report TR-2026-23, Institute for Physical AI @ JBI, July 2026. URL <https://physicalai-bmi.org/assets/papers/one-digest>. Self-published technical report, revised July 24, 2026.
- Chen, B., Wen, M., Shi, Y., Lin, D., Rajbahadur, G. K., and Jiang, Z. M. Towards training reproducible deep learning models. In *ICSE*, pp. 2202–2214, 2022. URL <https://arxiv.org/abs/2202.02326>.
- Chen, T. and Guestrin, C. XGBoost: A scalable tree boosting system. In *KDD*, pp. 785–794, 2016. doi: 10.1145/2939672.2939785.
- Chen, T.-R. The integer alibi: Localizing cross-kernel divergence in INT8-quantized LLM inference, 2026a. URL <https://arxiv.org/abs/2608.13756>. arXiv:2608.13756 [cs.LG].
- Chen, T.-R. Deterministic LLM inference across GPU kernels: Power-of-two INT8 quantization scales and the limits of tolerance-based conformance, 2026b. URL <https://arxiv.org/abs/2609.00363>. arXiv:2609.00363 [cs.LG].
- Collange, C., Defour, D., Graillat, S., and Iakymchuk, R. Numerical reproducibility for the parallel reduction on multi- and many-core architectures. *Parallel Computing*, 49:83–97, 2015. doi: 10.1016/j.parco.2015.09.001.
- Dawson, B. Floating-point determinism, 2013. URL <https://randomascii.wordpress.com/2013/07/16/floating-point-determinism/>.
- Demmel, J. and Nguyen, H. D. Fast reproducible floating-point summation. In *2013 IEEE 21st Symposium on Computer Arithmetic (ARITH)*, pp. 163–172, 2013. doi: 10.1109/ARITH.2013.9.
- Demmel, J., Ahrens, W., and Nguyen, H. D. Efficient reproducible floating point summation and BLAS. Technical Report UCB/EECS-2016-121, EECS Department, University of California, Berkeley, June 2016. URL <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-121.html>.
- Donaghy, J., Wilcox, B., Ersoy, O., Rastogi, S., St Arnaud, A., Titov, A., Greenberg, J., Fielding, B., and Grieve, H. OPEN-1B: A fully auditable training run, 2026. URL <https://arxiv.org/abs/2609.17380>. arXiv:2609.17380 [cs.LG], submitted September 15, 2026.
- Edwards, H. C., Trott, C. R., and Sunderland, D. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014. doi: 10.1016/j.jpdc.2014.07.003.
- Gensyn. RepOps, 2025. URL <https://github.com/gensyn-ai/repops-demo>. GitHub repository: bitwise reproducibility of ML operations across hardware targets.
- Gensyn. Introducing REE: Reproducible execution environment, March 2026. URL <https://www.gensyn.ai/news/ree>.
- Geurts, P., Ernst, D., and Wehenkel, L. Extremely randomized trees. *Machine Learning*, 63(1):3–42, 2006. doi: 10.1007/s10994-006-6226-1.
- Godoy, W. F., Melnichenko, T., Valero-Lara, P., Elwasif, W., Fackler, P., Ferreira Da Silva, R., Teranishi, K., and Vetter, J. S. Mojo: MLIR-based performance-portable HPC science kernels on GPUs for the Python ecosystem, 2025. URL <https://arxiv.org/abs/2509.21039>. arXiv:2509.21039 [cs.DC].
- Gond, R., Kamath, A. K., Ramjee, R., and Panwar, A. LLM-42: Enabling determinism in LLM inference with verified speculation, 2026. URL <https://arxiv.org/abs/2601.17768>. arXiv:2601.17768 [cs.LG].
- He, H. and Thinking Machines Lab. Defeating nondeterminism in LLM inference, 2025. URL <https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/>.
- Hendel, A. Mojolearn 162M: One model, two hardware routes. Hugging Face, 2026a. URL <https://huggingface.co/mojolearn-ai/mojolearn-162m>. Final checkpoint, tokenizer, training evidence and evaluations of the six-segment run.
- Hendel, A. mojolearn: Bitwise-identical GPU machine learning in Mojo on Apple Metal, NVIDIA CUDA and AMD HIP. Zenodo, 2026b. URL <https://doi.org/10.5281/zenodo.22864165>. Fixed evidence archive, evidence-0.8.9-cpu-gpu-20260920-r2.
- Hendel, A. mojolearn: Bitwise-identical GPU machine learning in Mojo on Apple Metal, NVIDIA CUDA and AMD HIP. Zenodo, 2026c. URL <https://doi.org/10.5281/zenodo.22675019>. Version 0.7.0 (alpha-api-0.7.0-20260909), software release archived September 9, 2026.

- Hendel, A. mojolearn: Bitwise-identical GPU machine learning in Mojo on Apple Metal, NVIDIA CUDA and AMD HIP. Zenodo, 2026d. URL <https://doi.org/10.5281/zenodo.22864816>. Version 0.8.10 (alpha-api-0.8.10-20260920), software release archived September 20, 2026.
- Heumos, L., Ehmele, P., Kuhn Cuellar, L., Menden, K., Miller, E., Lemke, S., Gabernet, G., and Nahnsen, S. mlf-core: A framework for deterministic machine learning. *Bioinformatics*, 39(4):btad164, 2023. doi: 10.1093/bioinformatics/btad164.
- Intel. Reproducibility conditions, 2024. URL <https://www.intel.com/content/www/us/en/docs/onemkl/developer-guide-windows/2024-2/reproducibility-conditions.html>. Developer Guide for Intel oneAPI Math Kernel Library, version 2024.2.
- Johnson, E., Lattner, C., Davis, T., Hoge, C., and Clayton, J. MAX is here! what does that mean for Mojo? Modular, February 2024. URL <https://www.modular.com/blog/mx-is-here-what-does-that-mean-for-mojo>.
- Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. LightGBM: A highly efficient gradient boosting decision tree. In *NeurIPS*, 2017.
- Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., and Zinenko, O. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14, 2021. doi: 10.1109/CGO51591.2021.9370308. URL <https://mlir.llvm.org/pubs/>.
- Li, R., Liu, L., Yang, G., Zhang, C., and Wang, B. Bitwise identical compiling setup: prospective for reproducibility and reliability of earth system modeling. *Geoscientific Model Development*, 9:731–748, 2016. doi: 10.5194/gmd-9-731-2016. URL <https://gmd.copernicus.org/articles/9/731/2016/>.
- LightGBM Developers. LightGBM installation guide, 2026. URL <https://lightgbm.readthedocs.io/en/stable/Installation-Guide.html>.
- Lubin, M. and Mueller-Albrecht, R. Floating point conditional numerical reproducibility on CPU and GPU with Intel oneAPI math kernel library (oneMKL). Intel developer article (archived), 2024. URL <https://www.intel.com/content/www/us/en/developer/archive/training/conditional-numerical-reproducibility-cn.html>. Updated March 25, 2024.
- Malyshau, D. Meganeura: Portable GPU training and inference through Vulkan and Metal, 2026. URL <https://arxiv.org/abs/2608.01563>. arXiv:2608.01563 [cs.LG].
- Modular. Mojo manual, 2026a. URL <https://mojolang.org/docs/manual/>.
- Modular. Mojo system requirements, 2026b. URL <https://mojolang.org/docs/requirements/>. Accessed September 8, 2026.
- NVIDIA. RAPIDS installation guide. NVIDIA RAPIDS documentation, 2026. URL <https://docs.nvidia.com/datascience/install/>. Requires an NVIDIA GPU with compute capability 7.0 or higher on Linux or WSL2; accessed September 20, 2026.
- OpenMD. Reproducibility in parallel OpenMD simulations, February 2013. URL <https://openmd.org/reproducibility-in-parallel-openmd-simulations/>.
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., VanderPlas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. Scikit-learn: Machine learning in Python. *JMLR*, 12: 2825–2830, 2011.
- Penedo, G., Kydlíček, H., Ben Allal, L., Lozhkov, A., Mitchell, M., Raffel, C., Von Werra, L., and Wolf, T. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024.
- Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. CatBoost: Unbiased boosting with categorical features. In *NeurIPS*, 2018.
- PyTorch. torch.use_deterministic_algorithms, 2026. URL https://pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html. PyTorch stable documentation; accessed September 20, 2026.
- RAPIDS Development Team. cuML: GPU machine learning algorithms, 2026. URL <https://github.com/NVIDIA/cuml/tree/265b9da6a0e75dbef071a3168398b993a5ff6f0e>. version 26.08 source and API documentation.
- Raschka, S., Patterson, J., and Nolet, C. Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2020. doi: 10.3390/info11040193.

- Rowley, A. MacBoost: Gradient boosted trees on Apple-silicon GPUs, 2026. URL <https://github.com/alex-rowley/macboost>. Software, version 1.2.0.
- Shi, Y., Ke, G., Chen, Z., Zheng, S., and Liu, T.-Y. Quantized training of gradient boosting decision trees. In *NeurIPS*, 2022.
- Srivastava, M., Arora, S., and Boneh, D. Optimistic verifiable training by controlling hardware nondeterminism. In *NeurIPS*, 2024. URL <https://arxiv.org/abs/2403.09603>. arXiv:2403.09603 [cs.CR].
- TensorFlow. tf.config.experimental.enable_op_determinism, 2024. URL https://www.tensorflow.org/api_docs/python/tf/config/experimental/enable_op_determinism. Page dated April 2024; accessed September 20, 2026.
- Tillet, P., Kung, H., and Cox, D. Triton: An intermediate language and compiler for tiled neural network computations. In *MAPL*, pp. 10–19, 2019. doi: 10.1145/3315508.3329973.
- vLLM Project. Batch invariance. vLLM documentation, 2026. URL https://docs.vllm.ai/en/latest/features/batch_invariance/.
- Wen, Z., Shi, J., He, B., Chen, J., Ramamohanarao, K., and Li, Q. Exploiting GPUs for efficient gradient boosting decision tree training. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2706–2717, 2019. doi: 10.1109/TPDS.2019.2920131.
- Wen, Z., Liu, H., Shi, J., Li, Q., He, B., and Chen, J. ThunderGBM: Fast GBDTs and random forests on GPUs. *JMLR*, 21(108):1–5, 2020.
- Whitehead, N. and Fit-Florea, A. Floating point and IEEE 754 compliance for NVIDIA GPUs. NVIDIA white paper, 2011. URL <https://docs.nvidia.com/cuda/floating-point/>. Continually updated documentation; accessed September 20, 2026.
- XGBoost Developers. XGBoost GPU support, 2026. URL <https://xgboost.readthedocs.io/en/stable/gpu/>.
- Xie, P., Zhang, X., and Chen, S. RepDL: Bit-level reproducible deep learning training and inference, 2025. URL <https://arxiv.org/abs/2510.09180>. arXiv:2510.09180 [cs.LG].
- yetalit. Mojmelo: Machine learning algorithms in pure Mojo. GitHub repository, 2024. URL <https://github.com/yetalit/Mojmelo>. accessed 2026-08-28.
- Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.
- Zhang, H., Si, S., and Hsieh, C.-J. GPU-acceleration for large-scale tree boosting. *arXiv:1706.08359*, 2017.
- Zhang, Z., Ding, X., Yuan, J., Liu, R., Mao, H., Xing, J., and Liu, Z. Deterministic inference across tensor parallel sizes that eliminates training-inference mismatch, 2025. URL <https://arxiv.org/abs/2511.17826v2>. arXiv:2511.17826, first posted November 21, 2025; version 2 revised May 29, 2026 [cs.LG].
- Zhu, Z., Thai, L., Yu, S., Liu, Y., Qiao, Y., Wang, C., Xu, H., and Shu, J. Demystifying numerical instability in LLM inference: Achieving reproducible inference for mission-critical tasks with HEAL, 2026. URL <https://arxiv.org/abs/2606.21023>. arXiv:2606.21023 [cs.LG].

A THE VERIFICATION RECORD

The release 0.8.10 record combines separate CPU, Apple Metal, NVIDIA CUDA and AMD HIP runs on fixed fixtures, and each underlying record retains its actual device, source and binary provenance. Of the 7,470 parts in the NVIDIA column, 5,427 were recorded on an H100, 1,611 on an RTX 4090 and 432 on an A100. In the AMD column, 5,616 were recorded on an MI325X and 1,854 on an MI300X. The Apple column is one M4. Of the 7,452 CPU parts, 6,103 were recorded on x86-64 servers, nearly all of them AMD EPYC processors of nine models, and 1,349 on the M4’s Arm cores. The record therefore spans more GPU models than the three named in the table headings: 972 parts also hold the same bits on a second NVIDIA model, and 616 on both AMD models. The reference table contains 273 configurations and 2,457 cells; the complete single-device subset is 214 configurations and 1,926 cells. Every one of its 7,452 CPU numerical parts has the same hash in all three GPU columns. The 18 additional GPU model-writer comparisons also agree. Non-applicability markers count as neither matches nor mismatches.

Devices outside the record. The six-segment training run (Section 7.2) adds NVIDIA L40S and Apple M3 Ultra GPUs, and the benchmark boards compare output hashes on the NVIDIA L40S, AMD MI325X and Apple M3 Ultra (Appendix D). Development and same-bits checks also ran extensively on Apple M2 Pro and M3 Ultra GPUs and on an NVIDIA RTX PRO 6000 Blackwell GPU.

The archived table decodes each device’s actual value, including explicit values that differ from a table entry’s default reference. The audit checks all applicable parts, not just training hashes. The final supplemental Apple, AMD and NVIDIA collections cover 342, 603 and 648 cells, respectively, with 1,647, 2,187 and 2,934 exact CPU-reference comparisons. These overlap the completed record and are not added to its totals. The supplemental collections fit each cell once; cross-device agreement does not require simultaneous runs, and a single fit does not establish within-device repeated-run stability.

The record is a composite, not one run of one binary. Its columns draw on records from 9 (AMD), 14 (NVIDIA), 21 (Apple) and 28 (CPU) source revisions committed between September 14 and 20, 2026. A record is admitted for a configuration only while that configuration’s numerical revision is unchanged, and each configuration carries that revision in the table. The archive also keeps the superseded records: 6,929 of their hash entries differ from the admitted values, and every one comes from a source revision older than the record admitted for that part. The completed reference table was archived after wheel publication. The archive also retains the exact-wheel publication and installed-check receipts; those packaging checks are distinct from the nu-

merical record. Appendix C identifies the fixed evidence archive.

A.1 The record by part

Table 5 counts the record by what was compared. The first three parts apply where the interface trains, predicts or serializes a model. The others apply where the interface has the property to test, and the verifier records explicit non-applicability where it does not.

A.2 The record by fixture

Each of the 214 single-device configurations is evaluated on every fixture of Table 7. The table counts numerical agreement across devices separately from whether a source record also repeated a fit.

fixture	what it attacks	parts	divergent
base	the control; ordinary dense input	830	0
ties	integer-grid data, so every distance and split value is tied many ways	830	0
hashed	values derived from a hash, with no structure, ties or symmetry	830	0
wide	column magnitudes from 10^{-4} to 10^4 , where accumulation order matters	830	0
denormal	part of the data below the smallest normal FP32 value	830	0
denormal, flushed	the same data with subnormals flushed to zero, the diagnostic twin	830	0
duplicates	duplicated rows, a constant column and an all-zero column	830	0
odd	12,345 rows and 17 features, which no block, warp or tile size divides	830	0
negative	every value negative, shifted off the origin	830	0

Table 7. The nine fixtures in the single-device record. Each fixture is run by all 214 configurations and contributes 830 matching numerical parts, with zero mismatches. Fixture inputs are identified by hash in the record, and a run whose input hash differs is rejected before any result is compared.

A.3 Negative controls

A negative control is a build of the library with one deliberate arithmetic change, run through the same verifier. It counts for a configuration only when a committed pair of runs, one clean and one changed, on the same device class, differ in at least one part of that configuration’s cells. Table 8 separates observed build changes, harness-only perturbations, declared controls and configurations without a control. The first category establishes sensitivity to changed

record group	configs	cells	parts	parts with equal bits		
				three GPU vendors	and CPU	divergent
gradient boosting	31	279	1,089	1,089	1,071	0
forests and isolation forest	13	117	459	459	459	0
clustering and mixtures	19	171	639	639	639	0
neighbors, indexes and density	25	225	909	909	909	0
linear models and SVM	24	216	837	837	837	0
kernel methods and Gaussian processes	20	180	702	702	702	0
decomposition and UMAP	5	45	189	189	189	0
time series	9	81	288	288	288	0
neural blocks	11	99	684	684	684	0
language-model training and inference	8	72	306	306	306	0
reduced-precision weights and matrix products	15	135	684	684	684	0
optimizers, losses, embeddings and tokenizers	13	117	261	261	261	0
linear algebra	6	54	99	99	99	0
metrics, preprocessing and model selection	15	135	324	324	324	0
total	214	1,926	7,470	7,470	7,452	0

Table 4. Single-device verification record for release 0.8.10. A part is one compared quantity of one cell, such as training state, saved-model bytes or held-out predictions. Every part has equal recorded bits across the Apple, NVIDIA and AMD columns, and all but 18 also across the CPU column; in those 18 saved-model comparisons the CPU loads the GPU-written file instead of writing its own (Table 5). The NVIDIA column is drawn from H100, RTX 4090 and A100 records and the AMD column from MI325X and MI300X records (Appendix A). The rows group configurations as the verifier records them, which splits some of the 12 algorithm families of Table 1. Counts exclude non-applicability markers and the 59 two-GPU configurations.

part	what is compared	configs	parts	parts with equal bits	
				three GPU vendors	and CPU
training	fitted state or the operation’s numerical result	214	1,926	1,926	1,926
saved model	serialized model bytes	148	1,332	1,332	1,314
inference	held-out predictions	183	1,647	1,647	1,647
batch invariance	outputs alone and under supported batch splits	182	1,638	1,638	1,638
serving-scale batch	1,024-row batches and sequence prefixes around chunk boundaries	29	261	261	261
ragged batches	padded sequences equal separate evaluation	20	180	180	180
decode equals forward	cached stepwise decoding equals full forward evaluation	18	162	162	162
gradient accumulation	per-row gradients and reduction-tree-preserving microbatch splits	16	144	144	144
sampler and trainer	sampled and recomputed log-probabilities	20	180	180	180
total			7,470	7,470	7,452

Table 5. Single-device record by numerical part. One part is one compared quantity of one cell, so a configuration contributes nine parts per row, one per fixture; N/A declarations are excluded. The 18 saved-model parts without a CPU value are the categorical and tensor CTR model tables on nine fixtures, where the CPU loads the GPU-written file and does not write a second model.

implementation arithmetic; harness perturbations establish sensitivity of the comparison.

negative-control evidence	single-device	two-GPU
changed build observed to change numerical bits	205	25
harness perturbation only	0	3
build control declared, no observed numerical change	9	0
no declared or observed control	0	31
total configurations	214	59

Table 8. Negative-control coverage in the archived inventory. Every single-device configuration has a declared control, and 205 of the 214 have been observed to move. A changed build and a harness perturbation establish different properties. An observed change on one fixture or part does not imply that every fixture or part is sensitive to that control.

tier	promise	what it pins
fast	none; speed only. The same fit on the same box may return different bits on two runs	nothing. Float atomics, hardware warp widths, vendor libraries, TF32 matrix product on NVIDIA
deterministic	same box, same build, same input gives the same bits, every run. Says nothing about a second box	every order decided at run time: float atomics, mutex merges, races. Keeps vendor arithmetic and the vendor matrix product
identical (default)	all of the above, and the same bits across Metal, CUDA and HIP	the above, plus denormal policy, FMA contraction, transcendentals, square root, and every geometry taken from a machine constant; closed vendor libraries never called

Table 6. The three numeric tiers. Each rung keeps the rung below it; one source, one flag, no fork.

Three controls illustrate what the comparison can see. Reversing the fold order of the language model’s output-head product changes 9 of 33 recorded losses on every CPU tested. Transposing one operand of the backward matrix product changes 8 of 10 recorded arrays and leaves all two losses equal, which is what corrupting the backward pass alone should produce, and the comparison names the first wrong tensor with both bit patterns. Zeroing the optimizer moments at a transferred checkpoint changes the following parameters and both moments while preserving the loss, gradients and token inputs. The property parts have controls of their own, which perturb one row of a whole-batch evaluation and must turn every batch-invariance cell from identical to divergent.

A.4 Two-GPU coverage and uncovered options

All 214 single-device configurations have complete numerical coverage across CPU and all three GPU columns. The 18 CPU saved-model writer exceptions are explicit roles, not missing predictions or failed training comparisons. The 59 two-GPU configurations are also complete across the three GPU vendors (Table 9): every numerical part has a value in the Apple, NVIDIA and AMD columns, and no part disagrees. These values were completed after the 0.8.10 release archive was frozen; they are the reference table at repository revision 47372db76, SHA-256 e95fbbb8...d0b7c9. The release archive itself holds 1,494 of the 1,935 parts complete across the three vendors.

two-GPU numerical-part coverage	parts
CPU and all three GPU columns agree	771
all three GPU columns agree; no numerical CPU role	1,164
incomplete GPU agreement	0
total numerical parts	1,935

Table 9. Coverage of the 59 two-GPU configurations on nine fixtures. Each vendor’s column runs the partitioned plan on one device. These parts are excluded from the single-device totals, and non-applicable roles are not counted.

The source inventory separately enumerates 255 public API

entries, including aliases, wrappers and helpers; every entry maps to a verifier configuration. That inventory is not a count of distinct algorithms, and mapping an entry does not establish complete coverage of its options. An interface can accept an option value that no recorded configuration exercises; such a call runs under the same contract but is not a verified result until a configuration records it.

What is counted as an algorithm. Table 1 groups the current algorithms into 13 families. An algorithm is a method a practitioner would name. A classifier and a regressor of the same method count once, as do its losses, kernels, distance metrics, penalties, starting rules and sharded two-GPU execution, all of which are configurations. The fixed release 0.8.10 reference table contains 273 configurations: 214 single-device configurations and 59 two-GPU configurations. Table 9 reports the latter’s coverage. The 24 evaluation metrics, such as accuracy, ROC AUC, silhouette and trustworthiness, are verified the same way and are not counted as algorithms.

A.5 Neural configurations and scale demonstrations

neural group	configs	cells	parts	divergent
transformer, full/windowed/decode	3	27	171	0
Mamba-1	2	18	99	0
Mamba-2	2	18	144	0
Mamba-3	1	9	72	0
hybrid stack	2	18	162	0
byte-level language model	5	45	252	0
MLP	1	9	36	0
BF16 and INT8 weight storage	12	108	648	0

Table 10. Selected neural configurations in the single-device record. Every listed part has equal bits across the Apple, NVIDIA, AMD and CPU columns on all nine fixtures. Reduced-precision weight configurations appear only in the last row. Property coverage is counted separately in Table 5.

Verifier fixtures are small so that anyone can run them. The

following training demonstrations provide separate checks beyond those fixtures.

A GPT-3 Small-shaped model. The decoder has 12 layers, width 768, 12 attention heads, context 2,048, a 50,257-token vocabulary and 162,147,840 parameters. Its blocks follow the Llama layer order (rotary positions, RMSNorm and a SwiGLU feed-forward of width 2,048), with untied input and output embeddings and no final norm; batch, peak learning rate and AdamW settings follow GPT-3 Small (Brown et al., 2020). The released final checkpoint has a held-out FineWeb-Edu loss of 3.27 nats per token (perplexity 26.2) and 29.6% zero-shot HellaSwag accuracy (Zellers et al., 2019) (length-normalized). GPT-3 Small reports 33.7% after 300 billion training tokens, about 114 times this run’s budget. Each optimizer step processes 64 logical shards of four sequences, or 524,288 tokens, using the same pinned FineWeb-Edu (Penedo et al., 2024) token stream and FP32 learning-rate table. Both routes start from the same step-0 checkpoint and continue from their own saved state. The state digest covers parameters, both AdamW moments and initialization flags; separate records bind the step counter, shard indices, recipe and token stream. Every step records the summed-gradient digest, 64 loss bit patterns and learning-rate bits as well.

segment	equal steps	divergent	files	endpoint SHA-256
1	1,000	0	11	ce8c6b98e45b
2	1,000	0	11	f03471f9afa9
3	400	0	5	82249517f68c
4	1,500	0	16	d935aba7cb29
5	1,000	0	11	8142c3eac17
6	100	0	2	4129921e99ed
total	5,000	0	56	

Table 11. Six-segment training evidence. Each equal step has matching state and summed-gradient digests, all 64 loss bit patterns, learning-rate bits and shard indices across routes. The files column counts matching checkpoint digests at common saved steps, excluding initialization. Endpoint digests are abbreviated for display only; all comparisons use full digests.

Handoff and simultaneous training. For the NVIDIA-to-Apple handoff into A/6, the M3 Ultra loads the H100’s step-4,898 checkpoint and independently recomputes steps 4,899 and 4,900. Both records match the sender’s state, gradient, loss and learning-rate fields. It then trains steps 4,901–5,000; the L40S independently computes those same 100 updates in B/6 and writes the same final checkpoint digest. The retained raw arrival records permit the same comparison for eight handoffs (16 replayed steps); the ledger additionally records PASS for the arrivals into A/4 and B/5. In the simultaneous A/3 segment, the H100 computes logical shards 0–43 and the MI325X computes shards 44–63. Both workers record equal state, gradient, loss and learning-rate fields

at every one of its 400 steps; B/3 independently obtains those values with one H100. Recovery overlaps within A/4 and B/5 are checked for equality and counted only once in Table 11.

control from step 100	verdict	changed fields
ordinary replay (H100, MI325X)	PASS	none
split-fold replay (H100)	PASS	none
zero optimizer moments	FAIL	state
63 rather than 64 shards	FAIL	state, grad., loss
swap shards 5 and 40	FAIL	state, grad., loss
swap shards 0 and 1	FAIL	loss only
one-ULP gradient edit (H100, MI325X)	FAIL	state, grad.

Table 12. Controls at the scale experiment’s full model shape. Passing runs reproduce steps 101 and 102; every deliberate change is detected at step 101. Loss denotes the ordered per-shard bit patterns.

A concrete failing record. In the H100 one-ULP control, element 123,585,811 of shard 63’s gradient changes from FP32 bits 3e4e221f to 3e4e2220. The summed element changes from 4139eddb to 4139eddc; at step 101 the state digest changes from abc8b816b5c3 to d3eefb6dba87 and the gradient digest from 25830bfc2016 to f386db390c34, while all 64 losses remain equal. The positive split-fold control exercises that same path without the edit and passes. Swapping only shards 0 and 1 is detected through loss ordering even though their commutative first addition leaves the summed gradient and updated state unchanged.

The evidence uses the recorded sliced-sha256-8.v2 digest scheme, which hashes all array bytes in eight contiguous ranges and combines their digests, and full SHA-256 for checkpoint files. The supplement retains the step chains, projected numerical and environment receipts, checkpoint digests, checkpoint-source records and control records with a file manifest and an evidence-file checker. It checks record completeness, agreement and checkpoint continuity without a GPU; it does not recompute training or independently validate the recorded GPU arithmetic. The run spans published wheels 0.8.15–0.8.24; the Apple segment uses 0.8.24. It is a separate scale record, not an extension of the release 0.8.10 fixture census.

A byte-level language model, step by step. A model with two decoder blocks and 34,944 parameters trains for 128 steps on pinned real text on all three vendors, and every recorded array of every step matches: parameters, gradients, AdamW moments, losses, held-out outputs and checkpoint bytes. The step-64 checkpoint resumes on the other vendor in both directions and rejoins the uninterrupted run. On seven CPUs with no GPU present, each given one step’s recorded state, all 640 recomputed gradients, losses, parameters and moments equal the GPU record.

One model trained on three vendors at once. The fixed fold that makes two-GPU training match one GPU also lets GPUs of different vendors, in different machines, train one model together. In the experimental `mojolearn.cross_vendor` interface, which is not part of release 0.8.10, each worker computes the gradients of the logical microbatches it owns, a coordinator sums all of them in microbatch order with the fold the GPU reduction uses, and every worker applies that sum with the same AdamW update. An Apple M4, an NVIDIA H100 and an AMD MI300X, each in its own machine, trained one decoder with 2 blocks, width 32, a 512-token vocabulary and 51,328 parameters for 6 steps, with four microbatches per step (two on the M4 and one on each of the other GPUs). After every step the three workers reported the same SHA-256 of the complete training state (parameters, both AdamW moments and flags), and each state equals the one a single GPU reaches by processing all four microbatches itself.

Runs of the same model in one process reach the same six states on one M4, one H100, two H100 GPUs and one MI300X. A checkpoint written after step 3 by the two H100 GPUs, and one written by the MI300X, each resume on the M4 and rejoin that trajectory. Folding the recorded microbatch gradients under every assignment of microbatches to the Apple, NVIDIA and AMD runs, 1,512 assignments over six steps, reproduces the device sum in every case. The coordinator stops the run when any two workers’ state hashes differ, and it refused a worker that changed a single element of the summed gradient by about 10^{-7} . Every step exchanges full gradients, so the result concerns identity, not throughput.

B THE IDENTICAL-MODE CONTRACT

B.1 Modes and vendor arithmetic

The fixture record measures identical mode. Deterministic mode promises repeatability on one device, while identical mode additionally fixes cross-vendor arithmetic. The unpinned k-NN example in Figure 1 illustrates how vendor square-root behavior can alter bits despite matching preceding operations. This record supplies no library-wide deterministic-mode mismatch rate.

B.2 Hazards by family

family	what can vary	effect on the model	identical-mode response
neural building blocks	matrix products, reductions and state updates	different activations, gradients or training state	portable primitives and explicit rounding barriers
trees and forests	histogram order, split-score rounding and ties	different split, partition or tree structure	fixed accumulation, frozen geometry and total tie rules
clustering and neighbors	distance reductions, square root, traversal and output order	different neighbors, memberships, clusters or labels	portable distance arithmetic and canonical ordering
linear and de-composition	matrix-product order, solver folds and closed vendor libraries	different coefficients, margins or components	portable matrix operations and fixed fold order
time series	recurrences, elementary functions and pivot ties	different likelihoods, parameters or forecasts	pinned functions, pivots and decision rules

Table 13. Algorithmic breadth exercises distinct numerical failure modes. The claim is one contract across these mechanisms, not novelty in the learning algorithms themselves.

B.3 Numerical mechanisms

Two boundaries matter when interpreting the record. First, floating-point contraction must be controlled per expression: the Mamba-3 state-rotation fixture requires explicit intermediate rounding barriers, not simply a global policy enabling FMA. Second, host arithmetic can choose device geometry. Host-side arithmetic that selects geometry must therefore obey the same numerical profile; device arithmetic tests alone do not establish this. The portable-arithmetic probes check elementary functions, contraction and subnormal handling at their stated inputs. They do not prove all possible inputs or a future compiler’s behavior.

ledger group	action	entries
order-dependent accumulation	REPLACE	5
a machine query reaching a float reduction	PIN	8
vendor-divergent elementary functions	REPLACE	9
code generation	PIN	1
the IEEE underspecification	PIN	2
ties resolved by arrival order	PIN	4
closed vendor libraries	REPLACE or REFUSE	5
host-side arithmetic that selects a device configuration	PIN	1
iteration counts and caps	PIN	2
random draws	PIN	1
a race, not an order	REPLACE	1
vendor-dependent output order	REPLACE or REFUSE	1
fixed-order floating-point folds	REPLACE	4
sign and extrema conventions	PIN	2
composed algorithms and interface checks	PIN / REPLACE / REFUSE	16
Total		62

Table 14. All 62 numerical-pathway entries in the ledger, each assigned to one group. An entry may describe an operation, a composed algorithm or an interface check; it is not necessarily one code location.

C ARTIFACT

Mojolearn is open source under Apache-2.0 at <https://github.com/mojolearn/mojolearn>. Release 0.8.10 is published on PyPI for macOS arm64 and Linux x86-64 and archived as (Hendel, 2026d), DOI 10.5281/zenodo.22864816; its reference table carries the single-device record unchanged. The numerical record is preserved in the fixed evidence archive (Hendel, 2026b), DOI 10.5281/zenodo.22864165. It includes the canonical table, the machine-readable audit, the raw-record manifest and all 548 referenced record entries, plus release receipts. The paper audit verifies the manifest hashes and the 29,862 recorded numerical column values underlying the single-device result. The project archive uses concept DOI 10.5281/zenodo.22068632.

The six-segment scale experiment has a separate supplement containing the complete per-step comparison records and an evidence-file checker (Appendix A.5). The final checkpoint is published with that supplement as a model repository, together with the tokenizer, a weights-only export, an off-line integrity verifier and held-out evaluations. The model repository is <https://huggingface.co/mojolearn-ai/mojolearn-162m> (Hendel, 2026a), DOI 10.57967/hf/10832; its source, with the supplement, is in the Mojolearn repository at <https://github.com/mojolearn/mojolearn/tree/main/release/gpt3-small-162m>.

Implementation size. Table 15 counts the source at revision 42d1e42c with cloc 2.06, code lines only, over every Git-tracked file in a programming language; data and prose files are excluded. The Mojo is divided by role. The library row holds the estimators, kernels and bindings. The verification row holds the host reference implementations the GPU kernels are compared against, the per-family checks, the fixtures and the sabotage arms that must fail before a result counts; 929 source files outside the result directories name a sabotage arm. Verification is 0.7 times the size of the library it checks. Python holds the public package, its tests and the gate, tooling and packaging scripts. The result directories retain another 10,190 lines of copied scripts beside the evidence they produced, and the table excludes them. Code size describes implementation scope, not numerical coverage or correctness.

language and role	files	code lines
Mojo library (estimators, kernels, bindings)	1,651	446,531
Mojo verification (oracles, checks, fixtures, sabotage arms)	869	294,660
Mojo benchmarks	61	17,012
Python package	162	72,146
Python tests	269	36,992
Python gates, tooling and packaging	999	184,932
Python benchmarks	92	11,891
Python reference scripts	159	16,221
shell	680	58,603
C/C++ and headers	23	3,237
YAML and TOML configuration	23	3,908
total	4,988	1,146,133

Table 15. Source by language and role at revision 42d1e42c. Code lines exclude comments and blanks. The Mojo rows sum to 758,203 and the Python rows to 322,182.

D MEASUREMENTS BY DEVICE AND MODE

This appendix lists the headline cells of the benchmark boards: identical mode on NVIDIA L40S (sm_89) and AMD Instinct MI325X (gfx942), and identical and fast mode on Apple M3 Ultra. A board races, on one machine, the Mojolearn GPU arm and each opponent on the same rows of the same dataset. The tables keep the fit cells of the flagship algorithm of each family on its board datasets (taxi and istella for tabular algorithms; the board’s own dataset for the neural and sequence lanes), with one row per algorithm and dataset showing the fastest GPU opponent that completed. The NVIDIA and AMD boards ran release 0.8.25 from September 29 to October 1, 2026; opponent-only boards on further machines with the same GPU models were recorded on October 6, 2026. The Apple board ran the 0.8.34 wheel from October 1, 2026, with identical and fast cells refreshed through October 5, 2026. Each arm runs one unscored warm-up and one timed round (one run per arm), with arms interleaved; opponents were scored once and stored. A time is the wall-clock median in milliseconds of the public call. The Mojolearn clock starts with the inputs on the host, includes the host-to-device copy and ends at a synchronized result; opponents whose clock starts with the inputs already on the device are marked [†]. Quality is the metric the board stores for each algorithm: held-out metrics for trees, and for the other families one float64 function over each arm’s outputs. One metric is shown per row, named in the first quality column. The ratio divides our median by the opponent’s; below 1.00 our median was the lower. A cell that did not complete shows its stored outcome (failed, refused, timeout, mode mismatch, out of memory, quality failed, unknown), and “–” marks a value the board does not hold. Cells the boards mark as superseded or as history are omitted. Opponent cells at chance-level quality (AUC at most 0.501) are broken runs; they are excluded from these tables and from the full record, and are listed with their stored values in the record’s last section (“Excluded opponent cells”). Source paths, hashes, dates and cell counts of the board files are in results/boards-20261008/*/PROVENANCE.md.

Output hashes on the boards. Separately from the release 0.8.10 record, benchmark boards race the Mojolearn GPU arm against opponents on one machine per device and store each cell’s time, quality and output hash. The NVIDIA L40S and AMD MI325X boards ran release 0.8.25 in identical mode. Of the 604 cells, in 202 configurations, that completed on both, 598 store the same output hash on the two vendors; the other six differ, in the three configurations kernel-shap, permutation-shap and quantile. The Apple M3 Ultra board ran release 0.8.34; bits may change between releases, so its column is recorded, not required to match. Of the 517 cells, in 184 configurations, that completed on all three boards, 389 store the same hash on all three, fewer than on NVIDIA and AMD alone: in 124 cells the M3 Ultra hash differs from an agreeing NVIDIA and AMD pair. Cells that completed cover 224 configurations on eight datasets on the L40S and 217 configurations on nine datasets on the MI325X; on the M3 Ultra, 244 configurations on nine datasets in identical mode and 243 configurations on eight datasets in fast mode.

Across all identical-mode fit cells, ours is the lower in 33 of 201 on the NVIDIA L40S (geometric mean of ours/theirs 7.20) and in 11 of 49 on the AMD MI325X (7.13). On the Apple M3 Ultra, identical mode is the lower in 16 of 60 (3.62) and fast mode in 16 of 52 (3.83).

D.1 NVIDIA L40S, IDENTICAL

Table 16: Headline measurements, NVIDIA L40S, IDENTICAL. Board of September 29 to October 1, 2026, release 0.8.25. Opponent tags: ^a the same board; ^b and ^c opponent-only boards of October 6, 2026 on two further L40S machines. Fit cells of the flagship algorithms on their board datasets; each row shows the fastest GPU opponent that completed (“–” where none completed). Times are medians in milliseconds; ours/theirs divides our median by the opponent’s. [†] the opponent’s clock starts with its inputs already on the device and excludes the host-to-device copy that our clock includes. [‡] the opponent’s model was fitted before its clock started. [§] the harness marked this comparison not comparable (the two models differ materially in size, for example in leaf count), so the ratio is not a like-for-like time. 69 rows.

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
trees	gbdt-ordered	istella	143,000	catboost-gpu 1.2. 10 ^a	38,100	3.77 [§]	auc 0.972	0.980
trees	gbdt-ordered	taxi	244,000	catboost-gpu 1.2. 10 ^b	20,300	12.05 [§]	auc 0.620	0.629
trees	gbdt-depthwise	istella	6,410	xgboost-gpu 3.2. 0 ^a	8,080	0.79 [§]	auc 0.979	0.984

Bitwise-Identical Machine Learning Across Metal, CUDA, and HIP

NVIDIA L40S, IDENTICAL, continued

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
trees	gbdt-depthwise	taxi	4,280	xgboost-gpu 3.2. 0 ^a	3,250	1.32 [§]	auc 0.625	0.631
trees	gbdt-lossguide	istella	21,000	xgboost-gpu 3.2. 0 ^a	12,000	1.75	auc 0.984	0.984
trees	gbdt-lossguide	taxi	8,050	xgboost-gpu 3.2. 0 ^a	6,550	1.23 [§]	auc 0.631	0.631
trees	gbdt-symmetric	istella	5,260	catboost-gpu 1.2. 10 ^a	7,580	0.69 [§]	auc 0.977	0.980
trees	gbdt-symmetric	taxi	3,140	catboost-gpu 1.2. 10 ^a	5,720	0.55	auc 0.621	0.630
trees	gbdt-categorical	taxi	15,300	xgboost-gpu 3.2. 0 ^a	13,600	1.13 [§]	auc 0.630	0.632
trees	rf	istella	5,050	cuml-rf-gpu 26.08. 00 ^c	24,000	0.21	auc 0.945	0.945
trees	rf	taxi	5,800	cuml-rf-gpu 26.08. 00 ^b	20,700	0.28	auc 0.618	0.618
trees	et	istella	3,760	—	—	—	auc 0.938	—
trees	et	taxi	2,520	—	—	—	auc 0.619	—
classical	kmeans	istella	801	cuml-gpu 26.08. 00 ^a	483 [†]	1.66	inertia 6.05e17	6.11e17
classical	kmeans	taxi	324	cuml-gpu 26.08. 00 ^c	307 [†]	1.06	inertia 3.09e8	3.06e8
classical2	gmm	istella	974	—	—	—	bic −3.85e7	—
classical2	gmm	taxi	181	—	—	—	bic −3.67e6	—
classical	dbscan	istella	542,000	cuml-gpu 26.08. 00 ^a	66,400 [†]	8.17	n_clusters 40,100	40,100
classical	dbscan	taxi	failed	cuml-gpu 26.08. 00 ^c	437,000 [†]	—	—	n_clusters 36.0
classical	hdbscan	istella	failed	cuml-gpu 26.08. 00 ^a	862 [†]	—	—	n_clusters 53.0
classical	hdbscan	taxi	failed	cuml-gpu 26.08. 00 ^c	369 [†]	—	—	n_clusters 159
classical	pca	istella	231	torch-gpu 2.13.0+ cu129 ^a	16.9 [†]	13.73	explained_variance_ ratio_sum 1.00	1.00
classical	pca	taxi	12.7	torch-gpu 2.13.0+ cu129 ^a	1.43 [†]	8.82	explained_variance_ ratio_sum 1.00	1.00
classical2	tsvd	istella	96.4	cuml-gpu 26.08. 00 ^b	35.3 [†]	2.73	explained_variance_ ratio_sum 1.00	1.00
classical2	tsvd	taxi	6.73	cuml-gpu 26.08. 00 ^c	5.93 [†]	1.14	explained_variance_ ratio_sum 1.00	1.00
algos	randomized-svd	istella	936	torch-gpu 2.13.0+ cu129 ^c	65.7 [†]	14.25	relative_ reconstruction_error 2.36e−4	2.36e−4
algos	randomized-svd	taxi	215	torch-gpu 2.13.0+ cu129 ^b	32.0 [†]	6.72	relative_ reconstruction_error 0.0272	0.0272
classical	ols	istella	911	torch-gpu-eigh 2. 13.0+cu129 ^b	19.8 [†]	45.91	r2 0.332	0.152
classical	ols	taxi	260	torch-gpu-eigh 2. 13.0+cu129 ^c	1.95 [†]	133.34	r2 0.909	0.909
classical2	ridge	istella	800	cuml-gpu 26.08. 00 ^b	48.2 [†]	16.58	r2 0.329	−0.251
classical2	ridge	taxi	77.0	cuml-gpu 26.08. 00 ^a	9.36 [†]	8.23	r2 0.909	0.909
classical2	lasso	istella	5,730	cuml-gpu 26.08. 00 ^b	935 [†]	6.13	r2 0.311	0.311
classical2	lasso	taxi	63.5	cuml-gpu 26.08. 00 ^a	7.08 [†]	8.97	r2 0.909	0.909

Bitwise-Identical Machine Learning Across Metal, CUDA, and HIP

NVIDIA L40S, IDENTICAL, continued

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
classical2	elasticnet	istella	2,320	cuml-gpu 26.08. 00 ^a	316 [†]	7.33	r2 0.261	0.261
classical2	elasticnet	taxi	58.8	cuml-gpu 26.08. 00 ^b	5.85 [†]	10.06	r2 0.907	0.907
classical2	logreg	istella	6,280	cuml-gpu 26.08. 00 ^b	1,850 [†]	3.39	accuracy 0.925	0.924
classical2	logreg	taxi	91.3	cuml-gpu 26.08. 00 ^c	9.73 [†]	9.38	accuracy 0.763	0.763
algos	sgd-clf	istella	615,000	cuml-gpu 26.08. 00 ^c	4,840	127.10	accuracy 0.901	0.809
algos	sgd-clf	taxi	128,000	cuml-gpu 26.08. 00 ^b	1,540	82.89	accuracy 0.766	0.703
classical	knn	istella	44.0	cuml-gpu 26.08. 00 ^a	73.2 [†]	0.60	recall_at_k 0.976	0.976
classical	knn	taxi	24.5	cuml-gpu 26.08. 00 ^c	33.3 ^{†‡}	0.74	recall_at_k 1.00	1.00
classical	svc	istella	36.5	cuml-gpu 26.08. 00 ^a	23.0 [†]	1.59	accuracy 0.922	0.922
classical	svc	taxi	649	cuml-gpu 26.08. 00 ^a	378 [†]	1.72	accuracy 0.767	0.767
classical2	svr	istella	65.8	cuml-gpu 26.08. 00 ^a	40.2 [†]	1.63	r2 0.318	0.318
classical2	svr	taxi	56.4	cuml-gpu 26.08. 00 ^c	42.2 [†]	1.34	r2 0.768	0.768
classical2	kernel-ridge	istella	4,910	cuml-gpu 26.08. 00 ^b	324 [†]	15.15	r2 0.385	0.385
classical2	kernel-ridge	taxi	4,910	cuml-gpu 26.08. 00 ^c	326 [†]	15.07	r2 0.727	0.727
algos	gaussian-nb	istella	1,360	cuml-gpu 26.08. 00 ^c	126	10.80	accuracy 0.877	0.877
algos	gaussian-nb	taxi	342	cuml-gpu 26.08. 00 ^b	22.3	15.35	accuracy 0.720	0.720
classical2	ivf	istella	526,000	cUvs-gpu 26.08. 01 ^b	302 [†]	1,740	–	recall_at_k 1.00
classical2	ivf	taxi	34,900	cUvs-gpu 26.08. 01 ^a	219 [†]	158.80	–	–
classical2	umap	istella	368	cuml-gpu 26.08. 00 ^b	502 [†]	0.73	trustworthiness.k15 0.980	0.980
classical2	umap	taxi	142	cuml-gpu 26.08. 00 ^c	249 [†]	0.57	trustworthiness.k15 0.990	0.992
algos	tsne	istella	4,220	cuml-gpu 26.08. 00 ^a	454	9.30	trustworthiness.k15 0.992	0.990
algos	tsne	taxi	2,690	cuml-gpu 26.08. 00 ^c	397	6.77	trustworthiness.k15 0.999	0.998
neural	transformer-forward	gaussian	6.51	torch-eager-bf16 2.13.0+cu129 ^c	3.43	1.90	–	–
neural	lm-train-step	bytes	46.5	torch-compile-bf16 2.13.0+cu129 ^b	10.8	4.32	loss_first_step 9.02	9.02
neural	mamba1-forward	gaussian	9.93	mamba-ssm-fp32 2.3.2.post1+ge9594ce1c732 ^a	3.51	2.83	–	max_abs_diff_vs_ours 1.19e−7
neural	mamba2-forward	gaussian	13.3	mamba-ssm-tf32 2.3.2.post1+ge9594ce1c732 ^a	2.91	4.58	–	max_abs_diff_vs_ours 5.13e−4
neural	mamba3-forward	gaussian	5.74	mamba-ssm-tf32 2.3.2.post1+ge9594ce1c732 ^a	4.10	1.40	–	max_abs_diff_vs_ours 0.00138

Bitwise-Identical Machine Learning Across Metal, CUDA, and HIP

NVIDIA L40S, IDENTICAL, continued

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
neural	samba-train-step	bytes	159	torch-compile-tf32 2.13.0+cu129 ^b	43.2	3.69	loss_first_step 5.64	5.64
neural	mlp-train-step	gaussian	2.45	torch-eager-fp32 2.13.0+cu129 ^b	1.47	1.67	loss_first_step 1.16	1.16
algos	mlp-clf	istella	4,000	–	–	–	accuracy 0.944	–
algos	mlp-clf	taxi	3,460	–	–	–	accuracy 0.768	–
algos	cnn-clf	synthetic	131	torch-eager-fp32 2.13.0+cu129 ^a	209 [†]	0.63	accuracy 1.00	1.00
algos	lstm-clf	synthetic	846	torch-eager-fp32 2.13.0+cu129 ^b	516 [†]	1.64	accuracy 0.969	0.969
algos	lstm-clf	taxi-hourly	846	torch-eager-fp32 2.13.0+cu129 ^a	532 [†]	1.59	accuracy 0.868	0.868
algos	gru-clf	synthetic	738	torch-compile-fp32 2.13.0+cu129 ^a	556 [†]	1.33	accuracy 0.972	0.972
algos	gru-clf	taxi-hourly	738	torch-eager-fp32 2.13.0+cu129 ^a	552 [†]	1.34	accuracy 0.866	0.866

D.2 AMD MI325X, IDENTICAL

Table 17: Headline measurements, AMD MI325X, IDENTICAL. Board of September 29 to 30, 2026, release 0.8.25. Opponent tags: ^a the same board; ^b an opponent-only board of October 6, 2026 on a further MI325X machine. Fit cells of the flagship algorithms on their board datasets; each row shows the fastest GPU opponent that completed (“–” where none completed). Times are medians in milliseconds; ours/theirs divides our median by the opponent’s. [†] the opponent’s clock starts with its inputs already on the device and excludes the host-to-device copy that our clock includes. [‡] the opponent’s model was fitted before its clock started. [§] the harness marked this comparison not comparable (the two models differ materially in size, for example in leaf count), so the ratio is not a like-for-like time. 69 rows.

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
trees	gbdt-ordered	istella	86,000	–	–	–	auc 0.972	–
trees	gbdt-ordered	taxi	135,000	–	–	–	auc 0.620	–
trees	gbdt-depthwise	istella	6,970	–	–	–	auc 0.979	–
trees	gbdt-depthwise	taxi	6,380	–	–	–	auc 0.625	–
trees	gbdt-lossguide	istella	25,500	–	–	–	auc 0.984	–
trees	gbdt-lossguide	taxi	13,000	–	–	–	auc 0.631	–
trees	gbdt-symmetric	istella	5,610	–	–	–	auc 0.977	–
trees	gbdt-symmetric	taxi	3,220	–	–	–	auc 0.621	–
trees	gbdt-categorical	taxi	24,400	–	–	–	auc 0.630	–
trees	rf	istella	5,110	–	–	–	auc 0.945	–
trees	rf	taxi	6,750	–	–	–	auc 0.618	–
trees	et	istella	1,200	–	–	–	auc 0.938	–
trees	et	taxi	889	–	–	–	auc 0.619	–
classical	kmeans	istella	736	torch-gpu 2.13.0+rocm7.1 ^a	1,880 [†]	0.39	inertia 6.05e17	5.99e17
classical	kmeans	taxi	396	torch-gpu 2.6.0+rocm6.4.1.git1ded221d ^b	538 [†]	0.74	inertia 3.09e8	3.06e8
classical2	gmm	istella	1,690	–	–	–	bic –3.85e7	–
classical2	gmm	taxi	303	–	–	–	bic –3.67e6	–
classical	dbscan	istella	1.14e6	–	–	–	n_clusters 40,100	–
classical	dbscan	taxi	failed	–	–	–	–	–
classical	hdbscan	istella	failed	–	–	–	–	–
classical	hdbscan	taxi	failed	–	–	–	–	–

Bitwise-Identical Machine Learning Across Metal, CUDA, and HIP

AMD MI325X, IDENTICAL, continued

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
classical	pca	istella	288	torch-gpu 2.13.0+ rocm7.1 ^a	9.29 [†]	30.96	explained_variance_ ratio_sum 1.00	1.00
classical	pca	taxi	20.3	torch-gpu 2.13.0+ rocm7.1 ^a	1.15 [†]	17.60	explained_variance_ ratio_sum 1.00	1.00
classical2	tsvd	istella	66.4	–	–	–	explained_variance_ ratio_sum 1.00	–
classical2	tsvd	taxi	14.2	–	–	–	explained_variance_ ratio_sum 1.00	–
algos	randomized-svd	istella	841	torch-gpu 2.13.0+ rocm7.1 ^a	207 [†]	4.07	relative_ reconstruction_error 2.36e−4	2.36e−4
algos	randomized-svd	taxi	250	torch-gpu 2.13.0+ rocm7.1 ^a	121 [†]	2.07	relative_ reconstruction_error 0.0272	0.0272
classical	ols	istella	1,250	torch-gpu-eigh 2. 13.0+rocm7.1 ^a	31.1 [†]	40.11	r2 0.332	0.152
classical	ols	taxi	171	torch-gpu-eigh 2. 13.0+rocm7.1 ^a	24.3 [†]	7.01	r2 0.909	0.909
classical2	ridge	istella	1,160	–	–	–	r2 0.329	–
classical2	ridge	taxi	54.8	–	–	–	r2 0.909	–
classical2	lasso	istella	4,640	–	–	–	r2 0.311	–
classical2	lasso	taxi	55.5	–	–	–	r2 0.909	–
classical2	elasticnet	istella	1,770	–	–	–	r2 0.261	–
classical2	elasticnet	taxi	58.6	–	–	–	r2 0.907	–
classical2	logreg	istella	11,500	–	–	–	accuracy 0.925	–
classical2	logreg	taxi	414	–	–	–	accuracy 0.763	–
algos	sgd-clf	istella	1.67e6	–	–	–	accuracy 0.901	–
algos	sgd-clf	taxi	607,000	–	–	–	accuracy 0.766	–
classical	knn	istella	1,180	torch-gpu 2.13.0+ rocm7.1 ^a	29.1 [†]	40.74	recall_at_k 0.976	0.980
classical	knn	taxi	1,280	torch-gpu 2.13.0+ rocm7.1 ^a	23.9 [†]	53.75	recall_at_k 1.00	1.00
classical	svc	istella	65.6	–	–	–	accuracy 0.922	–
classical	svc	taxi	1,450	–	–	–	accuracy 0.767	–
classical2	svr	istella	129	–	–	–	r2 0.318	–
classical2	svr	taxi	113	–	–	–	r2 0.768	–
classical2	kernel-ridge	istella	11,800	–	–	–	r2 0.385	–
classical2	kernel-ridge	taxi	12,100	–	–	–	r2 0.727	–
algos	gaussian-nb	istella	824	–	–	–	accuracy 0.877	–
algos	gaussian-nb	taxi	473	–	–	–	accuracy 0.720	–
classical2	ivf	istella	265,000	–	–	–	–	–
classical2	ivf	taxi	20,400	–	–	–	–	–
classical2	umap	istella	499	–	–	–	trustworthiness.k15 0.980	–
classical2	umap	taxi	188	–	–	–	trustworthiness.k15 0.990	–
algos	tsne	istella	5,840	–	–	–	trustworthiness.k15 0.992	–
algos	tsne	taxi	5,080	–	–	–	trustworthiness.k15 0.999	–
neural	transformer- forward	gaussian	14.4	torch-eager-bf16 2. 13.0+rocm7.1 ^a	1.54	9.35	–	max_abs_diff_ vs_ours 0.00182
neural	lm-train-step	bytes	124	torch-compile- bf16 2.6.0+rocm6. 4.1.git1ded221d ^b	8.18	15.10	loss_first_step 9.02	9.02
neural	mamba1-forward	gaussian	18.9	torch-eager-fp32 2. 6.0+rocm6.4.1. git1ded221d ^b	87.6	0.22	–	–
neural	mamba2-forward	gaussian	16.6	torch-eager-bf16 2. 13.0+rocm7.1 ^a	8.18	2.03	–	max_abs_diff_ vs_ours 0.00714

Bitwise-Identical Machine Learning Across Metal, CUDA, and HIP

AMD MI325X, IDENTICAL, continued

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
neural	mamba3-forward	gaussian	11.4	torch-compile-fp32 2.13.0+rocm7.1 ^a	6.48	1.76	–	max_abs_diff_vs_ours 4.77e–7
neural	samba-train-step	bytes	240	torch-compile-fp32 2.13.0+rocm7.1 ^a	30.4	7.88	loss_first_step	5.64 5.64
neural	mlp-train-step	gaussian	2.82	torch-eager-fp32 2.6.0+rocm6.4.1.git1ded221d ^b	1.33	2.12	loss_first_step	1.16 1.16
algos	mlp-clf	istella	10,200	–	–	–	accuracy	0.944 –
algos	mlp-clf	taxi	9,700	–	–	–	accuracy	0.768 –
algos	cnn-clf	synthetic	179	torch-eager-fp32 2.6.0+rocm6.4.1.git1ded221d ^b	173 [†]	1.04	accuracy	1.00 1.00
algos	lstm-clf	synthetic	1,030	torch-eager-bf16 2.6.0+rocm6.4.1.git1ded221d ^b	855 [†]	1.20	accuracy	0.969 0.969
algos	lstm-clf	taxi-hourly	1,030	torch-compile-fp32 2.6.0+rocm6.4.1.git1ded221d ^b	816 [†]	1.26	accuracy	0.868 0.868
algos	gru-clf	synthetic	919	torch-eager-fp32 2.13.0+rocm7.1 ^a	1,740 [†]	0.53	accuracy	0.972 0.972
algos	gru-clf	taxi-hourly	919	torch-eager-fp32 2.13.0+rocm7.1 ^a	1,690 [†]	0.54	accuracy	0.866 0.866

D.3 Apple M3 Ultra, IDENTICAL

Table 18: Headline measurements, Apple M3 Ultra, IDENTICAL. Board of October 1 to 5, 2026 (wheel 0.8.34; IDENTICAL cells refreshed through October 5). Fit cells of the flagship algorithms on their board datasets; each row shows the fastest GPU opponent that completed (“–” where none completed). Times are medians in milliseconds; ours/theirs divides our median by the opponent’s. [†] the opponent’s clock starts with its inputs already on the device and excludes the host-to-device copy that our clock includes. [‡] the opponent’s model was fitted before its clock started. [§] the harness marked this comparison not comparable (the two models differ materially in size, for example in leaf count), so the ratio is not a like-for-like time. 52 rows.

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
classical	kmeans	istella	2,180	torch-gpu 2.13.0	7,460 [†]	0.29	inertia	6.05e17 5.99e17
classical	kmeans	taxi	973	torch-gpu 2.13.0	2,320 [†]	0.42	inertia	3.09e8 3.06e8
classical2	gmm	istella	8,840	–	–	–	bic	–3.85e7 –
classical2	gmm	taxi	604	–	–	–	bic	–3.67e6 –
classical	hdbscan	istella	45,700	–	–	–	n_clusters	47.0 –
classical	hdbscan	taxi	3,850	–	–	–	n_clusters	161 –
classical	pca	istella	725	–	–	–	explained_variance_ratio_sum	1.00 –
classical	pca	taxi	87.9	–	–	–	explained_variance_ratio_sum	1.00 –
classical2	tsvd	istella	447	–	–	–	explained_variance_ratio_sum	1.00 –
classical2	tsvd	taxi	27.5	–	–	–	explained_variance_ratio_sum	1.00 –
algos	randomized-svd	istella	692	–	–	–	relative_reconstruction_error	2.36e–4 –

Bitwise-Identical Machine Learning Across Metal, CUDA, and HIP

Apple M3 Ultra, IDENTICAL, continued

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
algos	randomized-svd	taxi	161	–	–	–	relative_ reconstruction_error 0.0272	–
classical	ols	istella	1,280	–	–	–	r2 0.333	–
classical	ols	taxi	121	–	–	–	r2 0.909	–
classical2	ridge	istella	1,440	–	–	–	r2 0.329	–
classical2	ridge	taxi	49.7	–	–	–	r2 0.909	–
classical2	lasso	istella	8,500	–	–	–	r2 0.311	–
classical2	lasso	taxi	76.1	–	–	–	r2 0.909	–
classical2	elasticnet	istella	2,950	–	–	–	r2 0.261	–
classical2	elasticnet	taxi	68.5	–	–	–	r2 0.907	–
classical2	logreg	istella	2,810	–	–	–	accuracy 0.925	–
classical2	logreg	taxi	25.9	–	–	–	accuracy 0.763	–
classical	knn	istella	764	torch-gpu 2.13.0	687 [†]	1.11	recall_at_k 0.976	0.980
classical	knn	taxi	715	torch-gpu 2.13.0	651 [†]	1.10	recall_at_k 1.00	1.00
classical	svc	istella	190	–	–	–	accuracy 0.922	–
classical	svc	taxi	2,550	–	–	–	accuracy 0.767	–
classical2	svr	istella	203	–	–	–	r2 0.318	–
classical2	svr	taxi	205	–	–	–	r2 0.768	–
classical2	kernel-ridge	istella	646	–	–	–	r2 0.385	–
classical2	kernel-ridge	taxi	662	–	–	–	r2 0.727	–
algos	gaussian-nb	istella	148	–	–	–	accuracy 0.877	–
algos	gaussian-nb	taxi	19.1	–	–	–	accuracy 0.720	–
classical2	ivf	istella	5,000	–	–	–	–	–
classical2	ivf	taxi	555	–	–	–	–	–
classical2	umap	istella	1,760	–	–	–	trustworthiness_k15 0.980	–
classical2	umap	taxi	777	–	–	–	trustworthiness_k15 0.990	–
algos	tsne	istella	7,080	–	–	–	trustworthiness_k15 0.992	–
algos	tsne	taxi	6,160	–	–	–	trustworthiness_k15 0.999	–
neural	transformer-forward	gaussian	14.1	torch-compile-fp32 2.13.0	5.00	2.82	–	–
neural	lm-train-step	bytes	185	torch-compile-bf16 2.13.0	43.0	4.30	loss_first_step 9.02	9.02
neural	mamba1-forward	gaussian	21.2	torch-eager-fp32 2.13.0	93.3	0.23	–	–
neural	mamba2-forward	gaussian	33.9	torch-compile-bf16 2.13.0	154	0.22	–	–
neural	mamba3-forward	gaussian	29.0	torch-eager-fp32 2.13.0	80.7	0.36	–	–
neural	samba-train-step	bytes	339	torch-eager-fp32 2.13.0	103	3.28	loss_first_step 5.64	5.64
neural	mlp-train-step	gaussian	8.36	torch-compile-bf16 2.13.0	4.17	2.00	loss_first_step 1.16	1.16
algos	mlp-clf	istella	11,600	–	–	–	accuracy 0.944	–
algos	mlp-clf	taxi	8,720	–	–	–	accuracy 0.768	–
algos	cnn-clf	synthetic	1,110	torch-compile-bf16 2.13.0	286 [†]	3.86	accuracy 1.00	1.00
algos	lstm-clf	synthetic	1,990	torch-eager-fp32 2.13.0	727 [†]	2.74	accuracy 0.969	0.969
algos	lstm-clf	taxi-hourly	1,990	torch-compile-fp32 2.13.0	733 [†]	2.72	accuracy 0.868	0.868
algos	gru-clf	synthetic	1,730	torch-eager-fp32 2.13.0	3,160 [†]	0.55	accuracy 0.972	0.972
algos	gru-clf	taxi-hourly	1,720	torch-eager-fp32 2.13.0	3,130 [†]	0.55	accuracy 0.866	0.866

D.4 Apple M3 Ultra, FAST

Table 19: Headline measurements, Apple M3 Ultra, FAST. Board of October 1 to 5, 2026 (wheel 0.8.34; FAST cells refreshed through October 5). Fit cells of the flagship algorithms on their board datasets; each row shows the fastest GPU opponent that completed (“–” where none completed). Times are medians in milliseconds; ours/theirs divides our median by the opponent’s. [†] the opponent’s clock starts with its inputs already on the device and excludes the host-to-device copy that our clock includes. [‡] the opponent’s model was fitted before its clock started. [§] the harness marked this comparison not comparable (the two models differ materially in size, for example in leaf count), so the ratio is not a like-for-like time. 60 rows.

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
trees	gbdt-ordered	istella	63,600	–	–	–	auc 0.980	–
trees	gbdt-ordered	taxi	56,100	–	–	–	auc 0.628	–
trees	gbdt-depthwise	istella	16,600	–	–	–	auc 0.983	–
trees	gbdt-depthwise	taxi	10,800	–	–	–	auc 0.632	–
trees	gbdt-lossguide	istella	18,400	–	–	–	auc 0.984	–
trees	gbdt-lossguide	taxi	12,700	–	–	–	auc 0.632	–
trees	gbdt-symmetric	istella	14,300	–	–	–	auc 0.980	–
trees	gbdt-symmetric	taxi	11,300	–	–	–	auc 0.630	–
trees	gbdt-categorical	taxi	25,500	–	–	–	auc 0.631	–
trees	rf	istella	10,200	–	–	–	auc 0.945	–
trees	rf	taxi	10,700	–	–	–	auc 0.618	–
trees	et	istella	4,030	–	–	–	auc 0.938	–
trees	et	taxi	2,850	–	–	–	auc 0.619	–
classical	kmeans	istella	1,590	torch-gpu 2.13.0	7,460 [†]	0.21	inertia 6.05e17	5.99e17
classical	kmeans	taxi	1,010	torch-gpu 2.13.0	2,320 [†]	0.43	inertia 3.09e8	3.06e8
classical2	gmm	istella	5,860	–	–	–	bic –3.85e7	–
classical2	gmm	taxi	failed	–	–	–	–	–
classical	hdbscan	istella	3,800	–	–	–	n_clusters 47.0	–
classical	hdbscan	taxi	434	–	–	–	n_clusters 160	–
classical	pca	istella	218	–	–	–	explained_variance_	–
							ratio_sum 1.00	–
classical	pca	taxi	85.0	–	–	–	explained_variance_	–
							ratio_sum 1.00	–
classical2	tsvd	istella	921	–	–	–	relative_	–
							reconstruction_error	–
							1.22e–4	–
classical2	tsvd	taxi	27.8	–	–	–	relative_	–
							reconstruction_error	–
							0.00326	–
algos	randomized-svd	istella	435	–	–	–	relative_	–
							reconstruction_error	–
							2.36e–4	–
algos	randomized-svd	taxi	190	–	–	–	relative_	–
							reconstruction_error	–
							0.0272	–
classical	ols	istella	850	–	–	–	r2 0.332	–
classical	ols	taxi	106	–	–	–	r2 0.909	–
classical2	ridge	istella	538	–	–	–	r2 0.329	–
classical2	ridge	taxi	16.2	–	–	–	r2 0.909	–
classical2	lasso	istella	143	–	–	–	r2 0.311	–
classical2	lasso	taxi	12.8	–	–	–	r2 0.909	–
classical2	elasticnet	istella	150	–	–	–	r2 0.261	–
classical2	elasticnet	taxi	13.7	–	–	–	r2 0.907	–
classical2	logreg	istella	3,610	–	–	–	accuracy 0.925	–
classical2	logreg	taxi	32.7	–	–	–	accuracy 0.763	–
algos	sgd-clf	istella	2,960	–	–	–	accuracy 0.920	–
algos	sgd-clf	taxi	2,450	–	–	–	accuracy 0.755	–
classical	knn	istella	357	torch-gpu 2.13.0	687 [†]	0.52	recall_at_k 0.982	0.980

Bitwise-Identical Machine Learning Across Metal, CUDA, and HIP

Apple M3 Ultra, FAST, continued

family	algorithm	dataset	our ms	opponent	their ms	ours/ theirs	our quality	their quality
classical	knn	taxi	257	torch-gpu 2.13.0	651 [†]	0.39	recall_at_k 1.00	1.00
classical	svc	istella	168	–	–	–	accuracy 0.922	–
classical	svc	taxi	506	–	–	–	accuracy 0.767	–
classical2	svr	istella	182	–	–	–	r2 0.318	–
classical2	svr	taxi	99.3	–	–	–	r2 0.768	–
classical2	kernel-ridge	istella	298	–	–	–	r2 0.385	–
classical2	kernel-ridge	taxi	216	–	–	–	r2 0.727	–
algos	gaussian-nb	istella	123	–	–	–	accuracy 0.877	–
algos	gaussian-nb	taxi	25.4	–	–	–	accuracy 0.720	–
classical2	ivf	istella	1,740	–	–	–	–	–
classical2	ivf	taxi	188	–	–	–	–	–
classical2	umap	istella	929	–	–	–	trustworthiness.k15 0.982	–
classical2	umap	taxi	353	–	–	–	trustworthiness.k15 0.991	–
algos	tsne	istella	4,080	–	–	–	trustworthiness.k15 0.992	–
algos	tsne	taxi	3,260	–	–	–	trustworthiness.k15 0.999	–
algos	mlp-clf	istella	10,800	–	–	–	accuracy 0.944	–
algos	mlp-clf	taxi	8,050	–	–	–	accuracy 0.768	–
algos	cnn-clf	synthetic	1,080	torch-compile-bf16 2.13.0	286 [†]	3.79	accuracy 1.00	1.00
algos	lstm-clf	synthetic	1,880	torch-eager-fp32 2.13.0	727 [†]	2.59	accuracy 0.969	0.969
algos	lstm-clf	taxi-hourly	1,890	torch-compile-fp32 2.13.0	733 [†]	2.58	accuracy 0.868	0.868
algos	gru-clf	synthetic	1,650	torch-eager-fp32 2.13.0	3,160 [†]	0.52	accuracy 0.972	0.972
algos	gru-clf	taxi-hourly	1,650	torch-eager-fp32 2.13.0	3,130 [†]	0.53	accuracy 0.866	0.866

Every fit and inference cell of these boards, with every opponent, is in the full measurement record, [dist/measurements_full.pdf](#) (source [supplement/measurements_full_main.tex](#)), generated from the same board copies in [results/boards-20261008/](#).