Real Machine Learning on an AMD Radeon RX 6600: PyTorch, TensorFlow, and DirectML Without CUDA

Exzil Calanza

Independent Researcher, Iloilo City, Philippines
Independent research preprint. IEEE-style formatting only; this work has not been accepted, published, or peer reviewed by IEEE.

ABSTRACT This study evaluates whether a consumer AMD Radeon RX 6600 can execute practical machine-learning training and inference on Windows through Microsoft DirectML without CUDA, and whether GPU execution produces a meaningful performance or forecasting advantage under measured workloads. Live probes verified GPU execution for PyTorch through torch-directml, TensorFlow through the DirectML plugin, and ONNX Runtime through the DirectML execution provider. Warmed FP32 matrix multiplication achieved 1.75x, 5.26x, 8.04x, and 7.66x GPU speedups from 512 x 512 through 4096 x 4096 matrices, with approximately 2,710 GFLOPS measured on the RX 6600. A full MLP training loop reduced loss from 0.539 to 0.0003 over 300 steps and ran 1.84x faster than CPU, while a small graph remained slower because roughly 21 ms of dispatch overhead dominated. In the associated food-price experiment, an earlier LSTM failed out-of-time validation at 85.7% MAPE versus 7.41% for naive persistence. A DirectML-trained multi-horizon model reached 3.5% MAPE and 0.997 R-squared on 295 held-out points and beat persistence across eight origins, 4.65% versus 6.70% pooled MAPE over 2,360 points. The results show that DirectML can provide real AMD-GPU machine learning on Windows, but performance depends strongly on workload size and must be validated against task-specific baselines.

INDEX TERMS – AMD Radeon RX 6600; DirectML; GPU acceleration; machine learning; ONNX Runtime; PyTorch; time-series forecasting; Windows

I. INTRODUCTION

Most machine-learning tutorials quietly assume you own an NVIDIA card. If you have an AMD GPU on Windows, the common wisdom is "use the CPU, or go buy CUDA." This post is the counter-example. Every result below was produced on a **consumer AMD Radeon RX 6600** (gfx1032, 8 GB) on Windows 11 — training *and* inference, with mainstream Python ML frameworks, through Microsoft's **DirectML** backend.

To be clear about the goal: this is **general machine learning — training real models — not running a local chatbot.** The point is that PyTorch, TensorFlow, ONNX Runtime, and scikit-learn all put work on the AMD GPU, and that a model *trained on that GPU* now beats the honest baseline the project's old LSTM could not.

Every number here is backed by a reproducible JSON receipt; the driver, scripts, and receipts are open-source in the project repository (https://github.com/Zek21/ph-food-price-dashboard).

II. EXPERIMENTAL SETUP AND METHODOLOGY

A. The setup: one GPU, three Python environments

There is one wrinkle. `torch-directml` (PyTorch's DirectML backend) and the TensorFlow DirectML plugin do **not** ship wheels for Python 3.13, which was the only interpreter installed system-wide. The fix, with no administrator rights, is uv (https://github.com/astral-sh/uv) fetching standalone interpreters:

TABLE 1

Environment	Python	What runs on the AMD GPU
.venv-torch-dml	3.12	PyTorch 2.4.1 + torch-directml (train + infer)
.venv-directml	3.13	ONNX Runtime DirectML, scikit-learn → ONNX
.venv-tf-dml	3.10	TensorFlow 2.10 + DirectML plugin (train + infer)

III. DEVICE-PLACEMENT VALIDATION

B. Which ML libraries actually use the AMD GPU?

Each row below is a **live probe** on the RX 6600, not a marketing claim. For ONNX Runtime, GPU placement is confirmed by profiling per-node execution providers, not by a provider merely appearing in a list.

TABLE 2

Library	GPU on RX 6600	Mode	Notes
PyTorch (torch-directml)	✅ Yes	Train + infer	Full autograd on the privateuseone device
TensorFlow 2.10 (DirectML plugin)	✅ Yes	Train + infer	Keras fit() on /GPU:0
ONNX Runtime (DirectML EP)	✅ Yes	Infer	Any model exported to ONNX
CNNs (Conv2d / torchvision)	✅ Yes	Train + infer	via torch-directml
scikit-learn MLP → ONNX	✅ Yes	Infer	Gemm/MatMul run on DirectML
scikit-learn RandomForest → ONNX	⚠ CPU op	Infer	TreeEnsemble unimplemented on DirectML
PyTorch nn.LSTM	❌ No	—	Fused LSTM cell unimplemented on DirectML
LightGBM	❌ No	CPU	pip wheel has no OpenCL GPU build
XGBoost	❌ No	CPU	GPU path is CUDA/NVIDIA only
scikit-learn (native)	❌ No	CPU	No GPU backend (export to ONNX for the GPU)

Bottom line: the three big frameworks — PyTorch, TensorFlow, and ONNX Runtime — plus classic scikit-learn neural nets (once exported to ONNX) all accelerate on the AMD card. Only the gradient-boosting libraries stay on the CPU, because their GPU code targets CUDA or OpenCL rather than DirectML.

IV. COMPUTE PERFORMANCE RESULTS

C. Is the AMD GPU actually fast? PyTorch vs. CPU

Warmed FP32 matrix multiplication on the same machine, GPU (DirectML) vs. CPU:

TABLE 3

Matrix	GPU (DirectML)	CPU	GPU speedup
512 × 512	0.53 ms	0.93 ms	1.75×
1024 × 1024	1.25 ms	6.58 ms	5.26×
2048 × 2048	7.19 ms	57.86 ms	8.04×
4096 × 4096	50.72 ms	388.39 ms	7.66×

The RX 6600 reaches roughly **2,710 GFLOPS** FP32. A full MLP training loop ran end-to-end on the GPU — loss fell from 0.539 to 0.0003 over 300 steps, **1.84× faster than CPU**, with parameters and gradients verified to live on the DirectML device.

One honest caveat: for a *tiny* graph, DirectML is actually **slower** than the CPU, because a fixed ~21 ms GPU-dispatch overhead dwarfs a microsecond-scale model. The GPU wins once the work is genuinely ML-sized (≥ 1024^2 matmuls, real training loops). That regime is exactly what a proper GPU training path unlocks.

V. FORECASTING EVALUATION

D. Doing better: a GPU-trained forecaster that beats the honest baseline

The backdrop is a Philippine food-price forecasting project (https://exzilcalanza.info/philippine-food-price-prediction-machine-learning-wfp-2026/) built on 26 years of WFP data.

Its 2-layer LSTM failed an out-of-time validation badly: rolled forward from a January-2026 cutoff across February–June 2026, it scored **85.7% MAPE** with an R^2 of **−1.26** — worse than a flat line — while simply carrying the last observed price forward ("naive persistence") scored 7.4%. The forecasts were correctly withheld.

Because `torch-directml` can't run the fused LSTM cell, the model was rebuilt as a MatMul-based network that *does* train on the AMD GPU. Crucially, it predicts the price **change**, so "predict nothing" is exactly the naive baseline and any improvement is real, learned signal. It is judged on the **same 295 out-of-time points** the LSTM was.

Then I asked two independent AI advisors — **ChatGPT (GPT-5, "Sol")** and **Gemini 3.5**, over free browser sessions — how to push it further. They **converged** on the same plan: emit all horizons **directly** instead of rolling forward one step at a time (which accumulates error), use a **log-return target** against the persistence baseline, and train with a **horizon-weighted Huber** loss. Version 2 implements exactly that.

TABLE 4

Model (295 out-of-time points)	MAPE	MAE	R^2	Verdict
Old LSTM	85.7%	141.9	−1.26	❌ Withheld
Naive persistence (baseline)	7.41%	8.14	0.988	—

v1 GPU delta-MLP	~5.8%	~7.0	0.992	✅ Passed
v2 GPU multi-horizon	3.5%	4.5	0.997	✅ Passed

Version 2 roughly **halved** v1's error — and it holds up: tested across **eight** separate out-of-time origins, it **beat naive persistence on all 8** (pooled 4.65% vs 6.70% MAPE over 2,360 points). Each model trained on the RX 6600 in **three to four seconds**. That is the honest definition of "doing better": not a bigger number in a press release, but a model that clears a gate it previously failed, proven stable across many origins, on hardware most people already own.

VI. DISCUSSION AND EXPERIMENTAL FORECASTS

E. New forecasts (experimental)

Because v2 passes the out-of-time gate, its forward forecasts are publishable (unlike the withheld LSTM). A few examples, July 2026 → December 2027 (monthly average retail price, PHP):

TABLE 5

Commodity	Jul 2026	Dec 2026	Dec 2027
Eggs	9.06	9.33	9.89
Eggplant	82.14	104.40	113.63
Onions (red)	138.41	180.22	192.77
Onions (white)	140.47	164.68	171.75

Full forecasts for 59 commodities are in the repository. These are **experimental research outputs, not financial advice** — the model passed the out-of-time naive-baseline gate, but future accuracy is never guaranteed.

VII. REPRODUCIBILITY

F. Reproduce it

```
uv python install 3.12
uv venv --python 3.12 .venv-torch-dml
uv pip install --python .venv-torch-dml/Scripts/python.exe
torch-directml pandas

..venv-torch-dmlScriptspython.exe gpu_pytorch_proof.py
# PyTorch on the AMD GPU
..venv-torch-dmlScriptspython.exe gpu_forecaster_v2.py
# GPU-trained forecaster + gate
```

VIII. LIMITATIONS AND CONCLUSION

G. Truth boundary

These are warmed benchmarks on one host (AMD RX 6600) and one dataset. Passing the 2026 out-of-time gate — on eight origins — is strong evidence of skill on those windows, not a guarantee of future prices. "Trained on GPU" means the model's parameters and gradients lived on the DirectML device during training, verified in a hashed receipt. The natural next step, which both advisors also recommended, is to fold in the exogenous drivers already in the project — the ENSO climate index, the USD/PHP exchange rate, and the FAO Food Price Index.

The takeaway is simple: an ordinary AMD gaming GPU is a perfectly capable machine-learning device on Windows. You do not need CUDA to train real models — you need DirectML, the right framework, and honest validation.

IX. REFERENCES

1. [1] Microsoft, Get started with DirectML .
2. [2] Microsoft, Enable PyTorch with DirectML on Windows .
3. [3] Microsoft, Enable GPU acceleration for TensorFlow 2 with the DirectML plugin .
4. [4] ONNX Runtime, DirectML Execution Provider .
5. [5] E. Calanza, Philippine Food Price Dashboard reproducibility repository .

AUTHOR BIOGRAPHY

EXZIL CALANZA is an independent researcher and software builder based in Iloilo City, Philippines. His work focuses on browser and desktop automation, machine learning, reproducible evaluation, workflow verification, and AI-assisted systems engineering.