Empowering machine-learning assisted kernel decisions with eBPF^{ML}

Prabhpreet Singh Sodhi
Columbia University
pss2161@cs.columbia.edu

Georgios Liargkovas
Columbia University
g.liargkovas@cs.columbia.edu

Kostis Kaffes
Columbia University
kkaffes@cs.columbia.edu

ABSTRACT

Machine-learning (ML) techniques can optimize core operating system paths—scheduling, I/O, power, and memory—yet practical deployments remain rare. Existing prototypes either (i) bake simple heuristics directly into the kernel or (ii) off-load inference to user space to exploit discrete accelerators, both of which incur unacceptable engineering or latency cost. We argue that *eBPF*, the Linux kernel's safe, hot-swappable byte-code runtime, is the missing substrate for *moderately complex* in-kernel ML. We present *eBPF*^{ML}, a design that (1) extends the eBPF instruction set with matrix–multiply helpers, (2) leverages upcoming CPU matrix engines such as Intel Advanced Matrix Extensions (AMX) through the eBPF JIT, and (3) retains verifier guarantees and CO-RE portability.

CCS CONCEPTS

• **Software and its engineering → Operating systems**; • **Computing methodologies → Machine learning**.

KEYWORDS

Operating systems, eBPF, machine learning, hardware acceleration

ACM Reference Format:
Prabhpreet Singh Sodhi, Georgios Liargkovas, and Kostis Kaffes. 2025. Empowering machine-learning assisted kernel decisions with eBPF^{ML} . In *3rd Workshop on eBPF and Kernel Extensions (eBPF '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3748355.3748363

1 INTRODUCTION

Machine learning has transformed user-space systems; the next frontier is the operating-system (OS) kernel, where microsecond-scale decisions on CPU scheduling, disks, congestion control, or power state translate directly into latency and cost savings[1, 7, 9]. Unfortunately, ML-in-kernel prototypes to date remain special-purpose and fragile: (a) **Kernel rewrites.** Projects such as SelfTune [5] integrate bespoke ML logic deep in the scheduler, requiring years of main-line engineering. (b) **User-space offload.** Systems like LAKE [2] forward features to a GPU daemon, paying tens of microseconds in syscall and DMA overhead while competing with user workloads.

eBPF '25, September 8–11, 2025, Coimbra, Portugal

ACM ISBN 979-8-4007-2084-0/2025/09
https://doi.org/10.1145/3748355.3748363

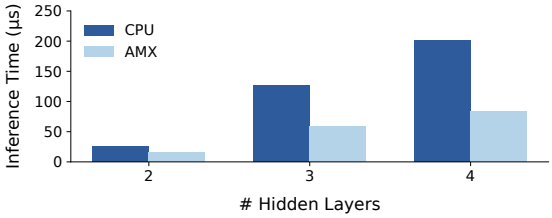

Figure 1: Per request inference times computed over 400k requests for the Heimdall [6] inference pipeline, an ML-based I/O admission policy for flash storage.

These approaches stall adoption: production kernels demand *(i)* minimal code churn, *(ii)* strict timing predictability, and *(iii)* architecture portability. We observe that eBPF already provides (i) and (iii) for non-ML extensions. The missing piece is performance: today's eBPF programs are limited to scalar integer/branch code because the verifier forbids loops and the JIT lacks vector or matrix primitives.

Key insight. Emerging *CPU matrix engines* (e.g., Intel AMX, ARM SME/SME2, RISC-V V) offer GPU-class throughput *inside* the CPU pipeline. Coupling these engines with eBPF's hot-load mechanism enables a sweet-spot: ML models that are richer than decision trees yet do not require a discrete accelerator or driver gymnastics. We therefore propose *eBPF*^{ML}, an architectural extension that allows kernel developers to attach pre-verified ML models as eBPF objects, automatically accelerated where hardware is present, and safely emulated elsewhere.

2 TO LEARN OR NOT TO LEARN

ML inside the kernel is attractive only when *(i)* the model's accuracy exceeds a heuristic by a margin that translates into user-visible gains *and (ii)* its inference latency does not dilute those gains. Prior studies illustrate both sides. Yawn [8] improves tail latency at web-scale *only* when request rates fall below 8k ops/s; at higher loads, idle-state exit time dominates. LAKE degrades NVMe throughput by 4–7% on light workloads because GPU batching fails to amortise PCIe round-trips [2]. These results suggest a moving inference threshold for general-purpose datacenter workloads. Complicating the picture is the diversity of ML models themselves. Techniques range from lightweight, single-tree classifiers to deep recurrent networks, each occupying a different spot on the accuracy vs. latency spectrum. A decision tree might execute in hundreds of nanoseconds but capture only coarse trends; a small multilayer perceptron (MLP) improves fidelity yet pushes inference into the few-microsecond regime; an LSTM or transformer offers still higher accuracy at the cost of tens of microseconds unless an accelerator is available. Because workloads, hardware, and latency budgets all

vary, *no single model can serve as the kernel's default brain.* Instead, any practical framework must let operators choose the **simplest** model that stays on the favorable side of the accuracy–latency tradeoff for their specific deployment, and it must do so without hard-coding that choice into kernel source.

3 HARDWARE CHOICES: CPU, GPU, OR INTEGRATED ACCELERATORS?

Discrete GPUs offer massive throughput but at DMA latency and require vendor driver hooks unavailable in the kernel. **Scalar CPUs** meet kernel safety requirements but cannot run even modest MLPs within the latency budget. In contrast, for moderately complex ML tasks, CPU-based accelerators like Intel's Advanced Matrix Extensions (AMX) [4] offer competitive performance to GPUs while avoiding the overhead associated with data transfers to discrete accelerators. In this context, a model's complexity can be defined not by its architecture or parameter count, but by a practical, system-level metric, its inference latency. If a model cannot execute within the kernel's microsecond-scale timing budget on a standard scalar CPU, we consider it "complex" enough to require acceleration. For example, we demonstrate in Figure 1 that AMX offers a promising alternative for fairly complex ML workloads, leading to 1.5-2.5x gains in inference time over CPU. Unless the workload is considerably complex, the high overhead of the GPU typically leads to performance degradation. For instance, in our measurements, the neural network inference time for LinnOS [3] was consistently higher on GPUs compared to AMX until the batch size of the input exceeded 700—a scenario not very relevant for practical applications.

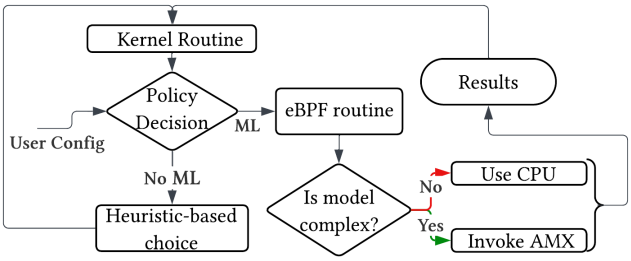

Figure 2: In-kernel ML inference decision flow using eBPF^{ML}

4 DESIGN VISION: TOWARDS EBPF-NATIVE ML

Our goal is *not* to prescribe a full upstream-ready implementation, but to outline a **plausible design envelope** that future kernel hackers can explore. The guiding constraint is to keep the familiar eBPF lifecycle—load, verify, attach, hot-swap—while making room for ML kernels whose inner loop is a matrix multiply.

Minimal, Generic Primitives: Rather than hard-code a specific accelerator or neural network layout, we conjecture that a *small set of generic linear-algebra helpers* is enough: matrix–multiply, bias-add, and a handful of element-wise activations. These could appear either as new eBPF opcodes or as helper calls; the exact choice is secondary, provided the verifier can ascribe a deterministic upper-bound on cycles and perform boundary checks on the data vectors for safety at each stage of the computation.

Hardware Adaptation Path: At load time the kernel identifies the most capable backend available on the host. If a matrix engine such as Intel AMX is present, the JIT lowers the generic primitive to a tiled kernel; otherwise it falls back to SIMD instruction sets like AVX-512 or even scalar code. This 'best-effort specialization' respects eBPF's CO-RE promise while allowing incremental adoption as hardware rolls out.

Verifier and Safety Questions: The current verifier approximates worst-case cycles statically. A matrix helper's latency, however, depends on data shape and backend. Can we extend the metadata model so that helpers annotate an upper-bound function of the dimensions of the input elements?

One other key consideration when targeting matrix accelerators like AMX is the management of its tile configuration and register state. The tile state is part of the processor's extended context managed via the XSAVE/XRSTOR mechanism, but it is not enabled by default in the Linux kernel. If AMX instructions are emitted by the JIT, the kernel must explicitly enable the XSAVE feature set for AMX and ensure the tile configuration and tile data are properly saved and restored across task switches within the kernel's FPU context. Without this explicit handling, improper context management can lead to data corruption, illegal instruction faults or system instability. Therefore, any AMX-aware JIT extension must either restrict usage to execution paths where AMX is used exclusively once across all execution levels and without preemption, or fully support safe and efficient context state management by enabling AMX XSAVE support and coordinating tile state during scheduling. The above sketch purposefully leaves ample room for exploration: How expressive should the algebra be? Where do we draw the line between verifier static checks and run-time accounting? What compiler pipeline emits BPF-friendly kernels? Our workshop submission aims to seed that discussion rather than settle it.

5 CONCLUSION

As computing systems grow increasingly complex—with on-core accelerators such as AMX, ever-changing workloads, and a rapid cadence of ML tooling—the kernel must evolve to support intelligent, real-time decision-making without compromising its reliability or maintainability. Our vision is a *modular, extensible* architecture in which eBPF remains the safe execution substrate while eBPF^{ML} supplies just enough linear-algebra primitives to embed learning-based logic directly inside critical kernel paths. Ultimately, we envision a kernel that is deeply integrated with modern ML workflows, bridging the gap between academic systems research and production deployment *without* rewriting core kernel logic.

ETHICAL CONSIDERATIONS

This work raises no ethical concerns.

REFERENCES

[1] Jingde Chen, Subho S Banerjee, Zbigniew T Kalbarczyk, and Ravishankar K Iyer. 2020. Machine learning for load balancing in the linux kernel. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*. 67–74.

[2] Henrique Fingler, Isha Tarte, Hangchen Yu, Ariel Szekely, Bodun Hu, Aditya Akella, and Christopher J Rossbach. 2023. Towards a Machine Learning-Assisted Kernel with LAKE. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 846–861.

[3] Mingzhe Hao, Levent Toksoz, Nanqinqin Li, Edward Edberg Halim, Henry Hoffmann, and Haryadi S Gunawi. 2020. {LinnOS}: Predictability on unpredictable flash storage with a light neural network. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 173–190.

[4] Intel Corporation. 2023. *Intel Advanced Matrix Extensions (Intel AMX) Technology Brief*.

[5] Ajaykrishna Karthikeyan, Nagarajan Natarajan, Gagan Somashekar, Lei Zhao, Ranjita Bhagwan, Rodrigo Fonseca, Tatiana Racheva, and Yogesh Bansal. 2023. {SelfTune}: Tuning Cluster Managers. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1097–1114.

[6] Daniar H. Kurniawan, Rani Ayu Putri, Peiran Qin, Kahfi S. Zulkifli, Ray A. O. Sinurat, Janki Bhimani, Sandeep Madireddy, Achmad Imam Kistijantoro, and Haryadi S. Gunawi. 2025. Heimdall: Optimizing Storage I/O Admission with Extensive Machine Learning Pipeline. In *Proceedings of the Twentieth European Conference on Computer Systems* (Rotterdam, Netherlands) *(EuroSys '25)*. Association for Computing Machinery, New York, NY, USA, 1109–1125. https://doi.org/10.1145/3689031.3717496

[7] Yu Liang, Riwei Pan, Tianyu Ren, Yufei Cui, Rachata Ausavarungnirun, Xianzhang Chen, Changlong Li, Tei-Wei Kuo, and Chun Jason Xue. 2022. {CacheSifter}: Sifting Cache Files for Boosted Mobile Performance and Lifetime. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*. 445–459.

[8] Erfan Sharafzadeh, Seyed Alireza Sanaee Kohroudi, Esmail Asyabi, and Mohsen Sharifi. 2019. Yawn: A CPU Idle-state Governor for Datacenter Applications. In *Proceedings of the 10th ACM SIGOPS Asia-Pacific Workshop on Systems* (Hangzhou, China) *(APSys '19)*. Association for Computing Machinery, New York, NY, USA, 91–98. https://doi.org/10.1145/3343737.3343740

[9] Yawen Wang, Kapil Arya, Marios Kogias, Manohar Vanga, Aditya Bhandari, Neeraja J Yadwadkar, Siddhartha Sen, Sameh Elnikety, Christos Kozyrakis, and Ricardo Bianchini. 2021. Smartharvest: Harvesting idle cpus safely and efficiently in the cloud. In *Proceedings of the Sixteenth European Conference on Computer Systems*. 1–16.