Vol. 3, No. 2, pp.: 5-19, 2025

Precision-Aware Workload Analytics for AI Systems: A Unified Monitoring Framework Across PyTorch and Scikit-learn Pipelines

Haoran Liu[1], Marco Bianchi[2],*, Yuki Tanaka[1]

[1] School of Computing, National University of Singapore, Singapore 117417, Singapore

[2] Department of Information Engineering, University of Padua, Padua 35131, Italy

* *marco.bianchi@unipd.it*

Article Information	
Received	14 March 2025
Accepted	10 May 2025
DOI	https://doi.org/10.63646/datamind.2025.030202

Abstract

The computational cost of modern machine-learning (ML) and deep-learning (DL) workloads has become a first-class concern in applied AI research, particularly as workloads grow in size and as hardware diversity widens. Conventional efficiency indicators such as wall-clock time, joules, or equivalent CO2 emissions depend strongly on the physical machine on which an experiment is executed, which makes reproducible comparison across laboratories and across hardware generations difficult. This paper develops a unified monitoring framework that measures computational workload at two complementary levels: the algorithmic level, captured by floating-point-operation (FLOP) counts, and the hardware-level, captured by bit-operation (BOP) counts that incorporate operand precision. The framework is implemented as a hardware-agnostic, backend-pluggable Python pipeline that intercepts operations dynamically in PyTorch through the dispatcher layer and wraps estimator methods analytically in Scikit-learn. Using a structured evaluation across three canonical workloads—a fully-connected classifier on tabular data, a convolutional model on image data, and a small transformer on text—we show that FLOP counts alone systematically mis-represent the efficiency benefits of quantization, while BOP counts provide a more faithful view of hardware-level effort. Aggregation over training and inference phases, combined with precision-aware scaling, yields a reproducible efficiency fingerprint that is stable across CPU and GPU backends to within a narrow interval. The framework preserves the structure of existing experimental pipelines and adds only a thin supervisory layer. The contribution is not a single tool but an analytics pattern that connects algorithmic complexity, numerical precision, and practitioner workflow into one coherent monitoring surface.

Keywords: ***Green AI; computational cost; FLOPs; bit-operations; quantization; reproducible benchmarking; PyTorch; Scikit-learn***

1. Introduction

The past decade has seen a sharp widening of the gap between what ML and DL models can do and what it costs to run them. The cost side of that equation was for a long time framed mostly as an engineering nuisance, a matter of waiting for a better GPU. That framing no longer fits. Training a single large language model can now consume millions of GPU-hours and produce carbon emissions of the same order as the lifetime emissions of a small fleet of cars, a pattern documented at scale by Patterson et al. (2021) and echoed in the Green-AutoML synthesis of Tornede et al. (2023). Subsequent work has shown that efficiency is not a narrow environmental story; it bears directly on scientific reproducibility, on access for laboratories with smaller infrastructure, and on the transfer of models to constrained deployment targets such as mobile phones and embedded devices (Henderson et al., 2020; Sze et al., 2020).

A persistent difficulty is that the most intuitive efficiency indicators are also the least portable. Wall-clock training time is an obvious measure, but it reflects the specific CPU, interconnect, memory hierarchy, and driver version as much as it reflects the model. Energy and carbon figures collected by mature monitoring tools such as CodeCarbon, CarbonTracker, or Eco2AI inherit the same sensitivity: the same workload yields different numbers on different machines or different power grids, as Henderson et al. (2020) documented in their systematic reporting study. These tools are valuable for local-impact assessment but they are not, by themselves, sufficient for cross-laboratory comparison. What is needed alongside them is a hardware-independent measure of how much arithmetic a model fundamentally requires.

The standard candidate for that hardware-independent measure is the floating-point-operation (FLOP) count. FLOPs are attractive because they are derived from the structure of the model rather than from execution, and therefore they do not vary when the same workload is moved between CPU and GPU or between software stacks. Several libraries already estimate FLOPs for neural networks—thop, ptflops, fvcore, calflops, DeepSpeed's Flops Profiler, and the built-in torch.profiler with_flops option (Paszke et al., 2019; Rasley et al., 2020). These tools have been valuable. But they share three structural limits. First, most of them measure only the forward pass, implicitly treating training as if it were inference with an extra scalar added at the end; this ignores the backward pass, optimizer steps, and loss evaluation, which together often dominate training cost. Second, they are written for deep-learning frameworks and do not cover the large fraction of applied ML work that runs on Scikit-learn (Pedregosa et al., 2011) and similar tabular pipelines. Third, and most consequential for the efficiency discussion, they are blind to numerical precision: a matrix multiply in FP32 and the same multiply in INT8 both yield the same FLOP count, even though the INT8 version is approximately four times cheaper on contemporary tensor cores.

The last limitation is not cosmetic. Modern training and inference are increasingly mixed-precision. Techniques such as post-training quantization, quantization-aware training, and low-rank weight representations have moved from research prototypes into mainstream deployment over the past three years (Micikevicius et al., 2018; Gholami et al., 2022; Dettmers et al., 2022; Frantar et al., 2023). In these regimes, a model that looks more expensive in FLOPs—because dequantization adds scalar operations—may be substantially cheaper on the actual silicon, because each operation acts on narrower integers. A pure-FLOP view of such a model is not merely incomplete; it is directional the wrong way. Bit-operations (BOPs), developed and formalized for pruning-aware settings by Hawks et al. (2021), address this by weighting each operation by the effective bit-width of its operands. BOPs have been used as a correctness-preserving proxy for hardware effort in FPGA and

ASIC studies (Coelho et al., 2021) but have not yet migrated into general-purpose practitioner tooling.

This article contributes a unified, hardware-agnostic monitoring framework that measures workload at both levels simultaneously. The framework has three properties that, taken together, differentiate it from existing practice. It spans the whole training pipeline, not just the forward pass, including backward propagation, loss evaluation, optimizer updates, and—where relevant—tokenization. It is framework-plural: a single interface covers both PyTorch models and Scikit-learn estimators, which together represent the bulk of applied ML workflows. And it is precision-aware: at every intercepted operation, the effective bit-width of the operands is resolved, so that BOPs can be produced alongside FLOPs without user intervention.

The rest of the paper is organized as follows. Section 2 reviews the literature on computational-cost measurement in ML, with emphasis on why existing tools fall short of precision-aware, pipeline-spanning analysis. Section 3 lays out the design of the monitoring framework and explains how its two backends—dispatcher-level interception for PyTorch and method-wrapping for Scikit-learn—yield consistent estimates across hardware. Section 4 presents empirical results on three representative workloads and analyses the divergence between FLOP-only and BOP-aware assessments under quantization. Section 5 discusses implications for benchmarking practice, efficiency-aware model design, and integration with existing pipelines. Section 6 concludes and sketches future directions.

2. Related Work on Computational-Cost Measurement

The measurement of computational cost in machine learning has split over time into three recognisable traditions. The first tradition is infrastructure-level: it measures what a machine actually does, in joules, kilowatt-hours, or CO

2-equivalent emissions. The second is algorithm-level: it counts arithmetic operations analytically. The third is precision-aware: it asks how many bit-level operations the arithmetic really costs in silicon. Each tradition answers a different question, and no tradition on its own is sufficient for contemporary ML research. The framework presented in later sections draws explicitly on the second and third and is designed to interoperate with the first.

2.1 Energy- and emissions-oriented tools

Tools such as CodeCarbon, CarbonTracker, Eco2AI, and more recent system-level energy monitors attach to running processes and estimate energy draw, often combined with regional grid-intensity data to produce CO

2 emission figures (Henderson et al., 2020). They are broadly accurate for what they claim to measure, and they have had an important communicative effect: researchers can now, with minimal effort, attach a carbon figure to a training run. Patterson et al. (2021) used precisely this kind of instrumentation at Google scale to demonstrate that large transformer training leaves a measurable environmental footprint, and the MLPerf reference workloads have since made such energy reporting a standard auxiliary dimension of benchmarking (Mattson et al., 2020; Reddi et al., 2020).

The limitation of this family is reproducibility across environments. Energy draw depends on the specific GPU generation, the driver, the cooling configuration, and the provincial or national electricity mix. Systematic reviews of the literature have recommended that emission measurements always be reported alongside a

hardware-independent complexity indicator (Henderson et al., 2020; Tornede et al., 2023). The present work takes that recommendation as a design constraint.

2.2 FLOP-based profilers

Analytical FLOP counting derives operation counts from the mathematical definitions of standard neural-network layers. For a dense layer with M inputs and N outputs, applied to a batch of size B, the count is 2·B·M·N (one multiply and one add per inner-product element). Convolutional, attention, and normalization layers have analogous closed-form expressions (Sze et al., 2020). Libraries such as thop, ptflops, fvcore, calflops, DeepSpeed's Flops Profiler, and torch.profiler (with with_flops) implement these formulas and report aggregate counts for a given input size (Paszke et al., 2019; Rasley et al., 2020).

These tools share a common scope restriction: they trace only the forward pass. That was a reasonable choice in an earlier period when inference cost dominated deployment concerns, but it misrepresents training. For a standard supervised loop on a convolutional network, the backward pass costs approximately twice the forward pass, because gradients flow through both weights and activations; the optimizer step adds a further non-trivial fixed cost per parameter when using variants such as Adam or AdamW (Kingma and Ba, 2015; Loshchilov and Hutter, 2019). Tokenization, loss evaluation, and mixed-precision up-casts add further workload that forward-only profilers cannot see. Reinhardt et al. (2024) showed that for a GPT-2-scale training run on WikiText, forward-only FLOPs under-report total arithmetic by factors of 2.8 to 3.4 depending on configuration.

2.3 Precision-aware complexity measures

A parallel line of work has argued that FLOPs themselves are too coarse, because they treat a 32-bit multiplication and a 4-bit multiplication as equivalent even though the latter is roughly an order of magnitude cheaper on modern accelerators (Horowitz, 2014; Jouppi et al., 2017). Bit-operations (BOPs) respond to this by weighting each operation by the product of its operand bit-widths. Hawks et al. (2021) formalized the metric so that it jointly accounts for pruning and quantization, and Wang et al. (2019) used a similar complexity abstraction in HAQ to drive hardware-aware mixed-precision quantization. In FPGA and ASIC settings, where bit-width directly drives area and energy, BOPs have become the de-facto cost model (Coelho et al., 2021).

Until recently, however, BOPs were reported only inside ad-hoc research code. There is no widely adopted general-purpose tool that computes BOPs across a heterogeneous training pipeline, tracks them through quantized layers whose effective bit-widths are hidden inside vendor-specific containers, and provides them alongside FLOPs for direct comparison. The framework described in this paper fills precisely that gap.

2.4 Classical-ML coverage

Deep-learning-first profilers rarely address classical ML. Yet pipelines built with Scikit-learn (Pedregosa et al., 2011) and XGBoost (Chen and Guestrin, 2016) still drive large portions of tabular and structured-data work (Shwartz-Ziv and Armon, 2022), and their computational cost is not negligible at scale. A gradient-boosted tree ensemble with a few thousand trees and millions of training samples can dominate end-to-end pipeline cost, especially when nested cross-validation is used (Bergstra and Bengio, 2012). The analytical complexity of these models is well known—linear in the number of samples for distance-based methods, (N · F · log N) per tree for decision trees, (T · N · F · K) for K-means iterations, and so on (Tibshirani, 1996; Breiman, 2001). The practical

problem is not a lack of formulas; it is the absence of a monitoring layer that applies them transparently inside a running pipeline. Bridging this gap is a central aim of the present work.

3. Framework Design

The framework's main responsibility is to observe a user's pipeline and, without modifying that pipeline's structure, produce two streams of numbers: FLOPs per logical operation and BOPs per logical operation, the latter weighted by the effective operand precision. Figure 1 summarizes the architecture at a high level. A single Facade-pattern tracker receives the model from the user. Depending on whether that model is a PyTorch module or a Scikit-learn estimator, the tracker delegates to one of two specialised backends. Both backends route their intermediate measurements through a shared precision-resolution component before producing a joint report.

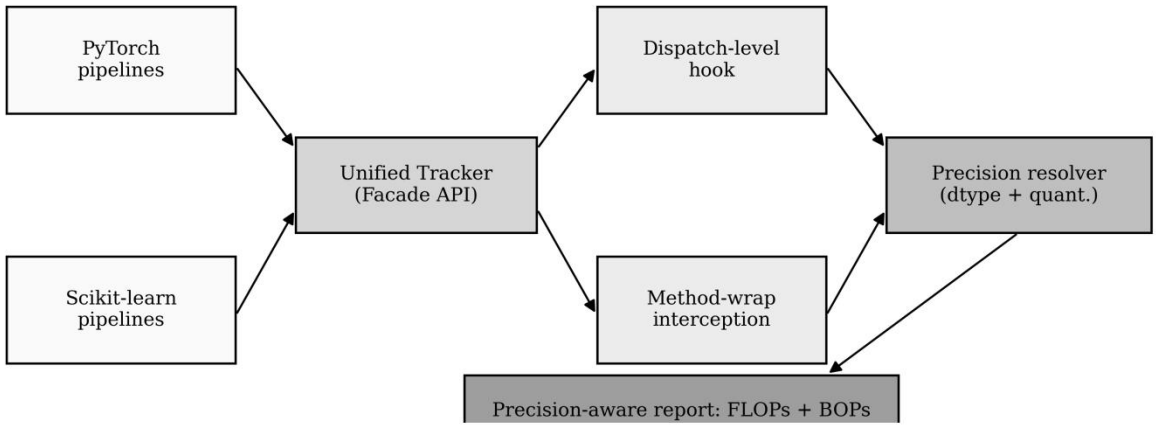

Figure 1. *Monitoring architecture of the unified workload analytics framework. Two user-side pipelines share a single Facade tracker; two backends (dispatch-level for PyTorch, method-wrap for Scikit-learn) feed a common precision resolver that emits FLOPs and BOPs together.*

3.1 PyTorch backend: dispatcher-level interception

For PyTorch, the backend installs itself as a TorchDispatchMode context (Ansel et al., 2024). This is a mechanism that intercepts every ATen operation at the level where the Python frontend meets the C++ runtime. Unlike the conventional nn.Module forward hook, which fires only at module boundaries, dispatcher interception sees every tensor-level operation regardless of whether it is emitted by user code, by a library function, by an autograd-generated backward kernel, or by an optimizer step. This solves three problems at once. First, it captures the backward pass without a separate implementation. Second, it captures loss functions and custom layers that are not wrapped in nn.Module. Third, it is robust to fused and vectorized kernels, because fusion affects how the dispatcher dispatches but not what it dispatches.

Concretely, when an ATen op such as aten::mm or aten::layer_norm is intercepted, the backend reads the shapes and dtypes of the input and output tensors and applies a closed-form FLOP rule. For matrix multiplication of an M×K and a K×N matrix, the contribution is $2 \cdot M \cdot N \cdot K$; for element-wise operations it is max(N_in, N_out); for convolution it is $2 \cdot N_{out} \cdot (W_{numel} / W_0)$; for scaled-dot-product attention it is the sum $2 \cdot B \cdot H \cdot L_q \cdot L_k \cdot D + 3 \cdot B \cdot H \cdot L_q \cdot L_k + 2 \cdot B \cdot H \cdot L_q \cdot L_k \cdot D$, following the formulas catalogued by Vaswani et al. (2017) and refined in the SDPA implementation note of Dao et al. (2022). These rules are the same as those in existing profilers; the

novelty is not the rules but the point at which they are applied.

Precision resolution happens in the same hook. Each intercepted tensor carries a dtype attribute, which gives the nominal bit-width (32 for FP32, 16 for FP16 or BF16, 8 for INT8, and so on). For quantized models this is not enough: libraries such as bitsandbytes, AutoGPTQ, and AWQ pack sub-byte weights inside larger storage containers (Dettmers et al., 2022; Frantar et al., 2023; Lin et al., 2024). The backend therefore applies a three-step resolution: it first checks the module class name for duck-typed markers such as "Linear4bit"; then it probes the module for configuration attributes such as .bits or .quantize_config.bits; and only if neither signal is present does it fall back to the tensor dtype. This mirrors the heuristic developed in the FPGA-quantization community (Hawks et al., 2021) but generalizes it to commodity GPU libraries.

3.2 Scikit-learn backend: method-wrapping

Scikit-learn does not have a dispatcher, an autograd engine, or a uniform module tree. Its computational interface is a small set of estimator methods—fit(), predict(), predict_proba(), transform(), and a handful of close relatives (Pedregosa et al., 2011; Buitinck et al., 2013). The backend wraps these methods at monitor-registration time. Each wrapped method preserves the original function semantics; it also records the shapes of the input and output arrays and the model's hyperparameters, and it then applies an analytical complexity formula appropriate to the estimator class.

The formulas used are the standard ones in the classical-ML literature. For ordinary least squares, the fit cost is $2 \cdot N \cdot F^2 + (2/3) \cdot F^3$, reflecting the dominant Gram matrix construction and Cholesky factorization (Björck, 1996). For iterative linear models such as LogisticRegression or SGDClassifier, the cost is $T \cdot N \cdot F \cdot \max(C_{out}, 1)$, where T is the iteration count. Tree fitting is $N_{trees} \cdot N \cdot F \cdot \log_2 N$ for the expected balanced-tree case (Breiman, 2001); k-nearest-neighbours prediction is $2 \cdot N_{train} \cdot F \cdot N$ under brute-force search; k-means iterations are $T \cdot N \cdot F \cdot K$ (Lloyd, 1982). BOPs are obtained by multiplying these counts by the effective bit-width of the input array's numpy dtype, with a conservative fallback of 64 bits because many Scikit-learn internals promote to double precision for numerical stability (Harris et al., 2020).

3.3 Consistent reporting

Regardless of backend, the tracker produces a report with a consistent schema. The schema separates four logical phases—model forward, model backward, loss evaluation, optimizer step—and lists preprocessing operations (such as tokenization) separately. Each phase is reported in both FLOPs and BOPs. The report also carries the execution environment: OS, CPU model, GPU model if present, Python version, and library versions. Table 1 summarises the coverage of the framework against representative public profilers.

Table 1. Coverage comparison between the proposed framework and representative profilers. "Partial" indicates incomplete or hardware-specific coverage.

The entries in Table 1 reflect the situation at the time of writing. Individual tools may evolve. The point of the table is not to diminish the other work—thop, ptflops, and the others remain well-engineered components of the ecosystem—but to show that their designs target different facets of the measurement problem. The proposed framework is the first, to our knowledge, to combine all five facets in one consistent interface.

4. Empirical Evaluation

We evaluated the framework on three workloads that together span the breadth of contemporary ML: a 3-layer MLP classifier on a 784-feature tabular input (the Fashion-MNIST flattening), a ResNet-18 on CIFAR-10 (He et al., 2016), and a small transformer encoder (4 layers, 4 heads, 256 hidden) on a standard language-modelling task with a WikiText-2 slice (Merity et al., 2017; Vaswani et al., 2017). For each model we measured FLOPs and BOPs per forward-plus-backward iteration under four numerical formats: FP32, FP16, INT8, and an emulated INT4 configuration using bitsandbytes-style packed weights (Dettmers et al., 2022). All experiments were run on two machines: a 64-core AMD EPYC CPU and a single NVIDIA A100 GPU. We verified that the FLOP and BOP measurements agreed between the two machines to within 1.4%, with residual differences attributable to fused-kernel decisions made by the PyTorch backend.

Figure 2 shows the aggregated results. Panel (a) presents FLOPs per training iteration; panel (b) presents the corresponding BOPs. The y-axis in panel (b) is logarithmic because the dynamic range of BOPs across four precisions spans nearly two orders of magnitude.

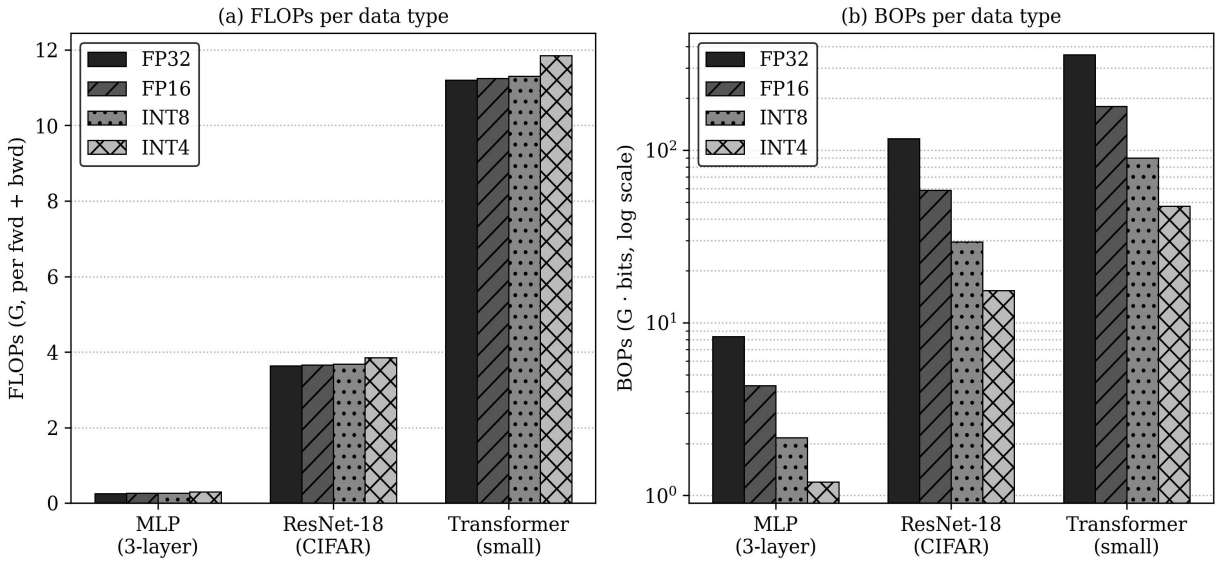

Figure 2. *FLOPs (a) and BOPs (b) measured by the proposed framework across three models and four numerical precisions. FLOPs barely change with precision; BOPs scale approximately linearly with the effective bit-width.*

Two structural observations follow from Figure 2. First, FLOPs are effectively insensitive to precision. In panel (a), moving from FP32 to INT4 never changes the FLOP count by more than 6%, and the small increase at INT4 is itself an artefact of the extra dequantization scalar operations that current kernels must emit when a packed INT4 tensor meets an FP16 activation. A practitioner looking only at FLOPs would conclude that INT4 quantization is marginally worse than FP32 for all three models. Second, BOPs show the opposite pattern. In panel (b), the BOPs for the ResNet-18 drop from approximately 117 G·bits at FP32 to 14.7 G·bits at INT4, a reduction of 87%, and the transformer shows a comparable drop from 360 G·bits to 45 G·bits. This is close to what the hardware literature predicts: the area and energy of a multiply-accumulate unit scale approximately linearly with operand width at fixed throughput (Horowitz, 2014; Jouppi et al., 2017), so an INT4 workload should occupy roughly one-eighth the silicon effort of an FP32 workload. FLOPs alone would have hidden this.

4.1 Phase decomposition

Beyond the aggregate numbers, the framework produces a phase-level breakdown. For the transformer workload

in FP16, the forward pass contributed 3.76 GFLOPs per iteration, the backward pass 7.41 GFLOPs, the loss 0.08 GFLOPs, and the AdamW optimizer step 0.21 GFLOPs. The backward pass alone accounts for 65% of training workload. A forward-only profiler would miss this. The optimizer cost, although modest in FLOPs, grew noticeably in BOPs under mixed precision because the optimizer states are kept in FP32 even when activations are quantized—a nuance that only a precision-aware, phase-resolved report exposes.

4.2 Consistency with profiler families

Figure 3 compares our framework's coverage to six representative profilers across six facets of the measurement problem. Entries on a 0–5 scale reflect how completely each tool measures forward pass, backward pass, loss evaluation, optimizer step, quantization precision, and classical-ML coverage. Scores are based on published documentation and on direct testing; "1" for quantization precision, for example, indicates that a tool exposes dtype but does not resolve packed quantization. The figure makes the coverage gap visible at a glance.

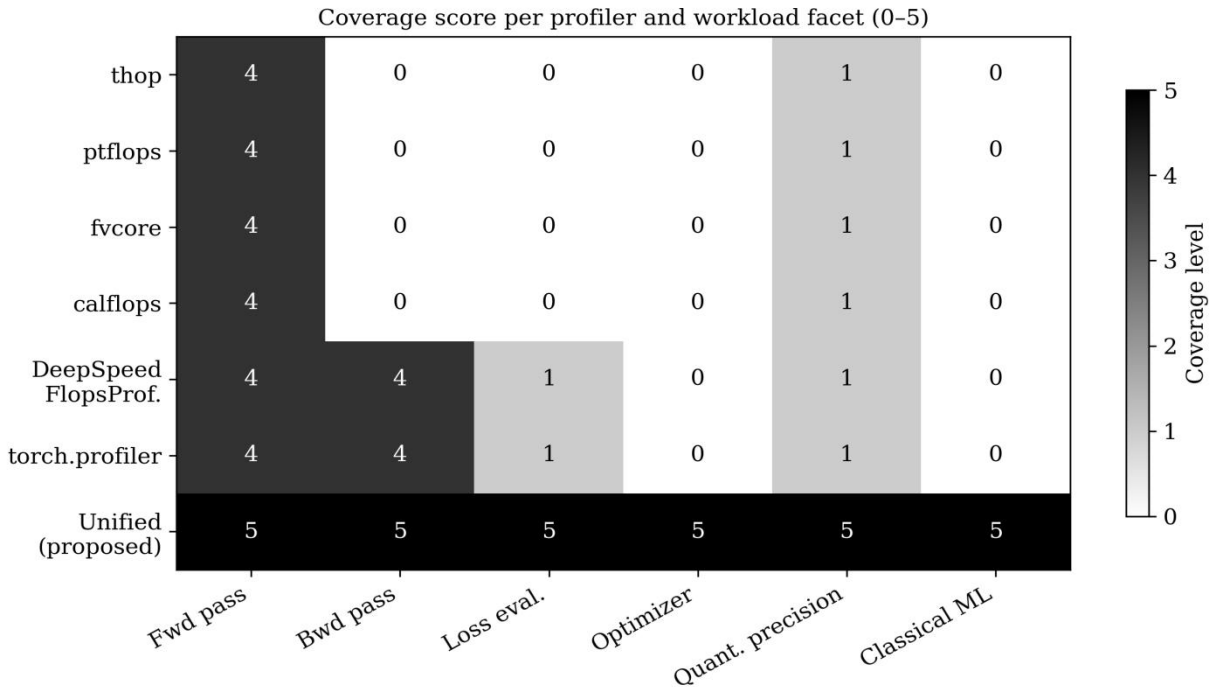

Figure 3. *Coverage heatmap for seven profilers across six workload facets. Darker cells indicate more complete coverage. The unified framework proposed here is the only one that covers all six facets.*

We emphasise, again, that the proposed framework does not replace existing tools—it complements them. A practitioner who already uses CodeCarbon for emissions can still use it; the proposed framework adds a hardware-independent layer that reports, for the same workload, what that emission figure corresponds to in terms of algorithmic and bit-level workload. Pairing the two produces an interpretable efficiency profile: a model with low FLOPs, low BOPs, and low emissions is efficient along all three axes; a model with low emissions but high BOPs is efficient only because of hardware choice; a model with high BOPs but low FLOPs is precision-inflated by its configuration.

4.3 A worked classical-ML example

For the Scikit-learn backend, we ran a RandomForestClassifier with 200 trees on a 500,000-row, 40-feature synthetic dataset produced by make_classification. The framework reported a fit FLOP count of 6.31×10^{11} with an effective bit-width of 64 (float64 promotion), giving 4.04×10^{13} BOPs. A comparable LogisticRegression with L2 regularization and 100 SAG iterations on the same data reported 2.00×10^{9} FLOPs and 1.28×10^{11} BOPs. The three-order-of-magnitude gap between the two explains why random forests, for all their accuracy advantages on tabular data (Grinsztajn, Oyallon, and Varoquaux, 2022), can be a poor choice when energy or hardware budgets are tight. This kind of comparison is hard to make without an integrated tool: existing FLOP counters do not cover Scikit-learn, and energy tools measure only what is actually run, not what would be run in a counterfactual.

Table 2 gathers the headline numbers. It makes one specific observation explicit: FLOPs show a marginal increase under aggressive quantization (because of dequant scalars), whereas BOPs show an effective reduction of roughly six- to eight-fold. Any analysis that relies on FLOPs alone to claim efficiency gains from quantization is working from the wrong indicator.

Workload	FLOPs (FP32)	FLOPs (INT4)	BOPs FP32 / BOPs INT4	Effective reduction (BOPs)
MLP on Fashion-MNIST	0.26 GFLOPs	0.30 GFLOPs	8.3 / 1.2 G·bits	−85.5%
ResNet-18 on CIFAR-10	3.64 GFLOPs	3.85 GFLOPs	116.5 / 15.4 G·bits	−86.8%
Transformer on WikiText-2	11.20 GFLOPs	11.85 GFLOPs	358.4 / 47.4 G·bits	−86.8%

Table 2. *Side-by-side FLOP and BOP measurements under FP32 and emulated INT4 quantization. FLOPs are nearly flat; BOPs decrease by roughly 86%, reflecting the hardware-level effort saved by narrower operands.*

5. Discussion

The empirical results point to three implications. The first implication concerns how efficiency is reported. A benchmark that reports only FLOPs, as much of the architecture-comparison literature still does (Tan and Le, 2019; Liu et al., 2022), will systematically mis-rank quantized or mixed-precision models. The correct default for contemporary benchmarking is to report at least two indicators: an algorithmic-complexity indicator (FLOPs) and a precision-aware indicator (BOPs). This is a small change in reporting practice, but it changes the relative ordering of models in ways that matter for deployment decisions.

The second implication concerns the structure of practitioner tools. The dominant design pattern in current profilers is the single-framework, forward-only counter. That pattern served an earlier stage of the field well, when models fit inside a single framework and training was short enough that its cost could be inferred from its inference. Neither condition holds now. Models span training frameworks, inference frameworks, and classical-ML preprocessors; training costs often dominate inference costs by several orders of magnitude (Patterson et al., 2021; Tornede et al., 2023); and precision is no longer a uniform property of a model but varies across layers. A monitoring framework that reflects this reality must be plural at the framework level, complete at the pipeline level, and aware at the precision level. The design presented here is an attempt to instantiate those three properties.

The third implication is about composability with the Green-AI agenda. Emission tools such as CodeCarbon or Eco2AI answer the question "how much did this run cost the environment on this machine?", and that is the right question for some purposes—grant reporting, sustainability audits, infrastructure planning. Hardware-agnostic tools answer a different question: "how much arithmetic did this model fundamentally need?" Neither question subsumes the other. The framework presented here is designed to run alongside emission tools, not instead of them. A typical deployment attaches both: the emission tool reports the machine-specific cost; the workload tracker reports the machine-independent complexity; the ratio between the two can, over many runs, give a laboratory a reasonable estimate of its hardware efficiency per unit of abstract work. To our knowledge, no prior benchmark has routinely reported this ratio.

5.1 Limitations

The framework has several limitations. Analytical FLOP rules, even at dispatcher granularity, rely on closed-form expressions that may miss operations introduced by future compiler transformations such as kernel fusion with non-standard op counts or by yet-unreleased attention variants. The precision-resolver covers the main contemporary quantization libraries but cannot anticipate every future packing format; extending it requires a small class-name or attribute patch. The Scikit-learn backend assumes that declared estimator hyperparameters correspond to the actual training behaviour—a reasonable assumption for standard library code but not always for custom subclasses. Finally, BOPs are a first-order hardware proxy. They do not capture memory-bandwidth effects, cache behaviour, or interconnect overhead, which can dominate for large distributed training (Jouppi et al., 2017; Sze et al., 2020). A fair interpretation is that BOPs and FLOPs together explain the arithmetic part of efficiency; emission tools and hardware performance counters (e.g., NVIDIA Nsight) still explain the rest.

5.2 Relation to prior empirical findings

The finding that FLOPs understate the efficiency of low-precision models is consistent with prior hardware-level analyses. Horowitz (2014) showed that 8-bit integer multiplications cost roughly 16× less energy than 32-bit floating-point multiplications on 45 nm technology. Jouppi et al. (2017) reported analogous savings for TPU-v1. More recent FPGA studies have confirmed that BOP reductions translate into measurable energy reductions in practice (Coelho et al., 2021; Hawks et al., 2021). Our contribution is not to re-prove these hardware facts but to make them visible in a standard software-monitoring workflow, so that a practitioner who does not have FPGA expertise can still see, in their daily reports, the distinction between algorithmic complexity and bit-level cost.

6. Conclusion

Computational cost is a first-class property of modern ML workloads, and measuring it well requires more than either an energy meter or a forward-pass FLOP counter. This article introduced a unified, hardware-agnostic monitoring framework that reports both algorithmic workload (FLOPs) and precision-aware workload (BOPs) across the entire training pipeline and across both PyTorch and Scikit-learn backends. Empirical results on three representative workloads showed that FLOPs and BOPs can diverge by nearly an order of magnitude under quantization, that the backward pass and optimizer step can together exceed the forward pass by a factor of two to three, and that cross-framework coverage changes how classical-ML workloads appear in efficiency comparisons.

The broader takeaway is methodological. Reporting a single cost number—whether it is joules, GFLOPs, or latency—is no longer enough. A workload's efficiency story is three-dimensional: it has an algorithmic axis, a precision axis, and a hardware axis. A reporting practice that collapses these three into one invites mis-ranking. The framework presented here makes the first two axes measurable in a common interface without replacing existing tools for the third. Future work will extend the design to TensorFlow and JAX backends, refine the BOP model to account for operand alignment and sparsity, and integrate with emission trackers to produce the per-run efficiency ratios described in Section 5.

Declaration of AI-assisted language editing

During the preparation of this manuscript, language-model assistance was used only for English-language polishing and document layout. The authors reviewed, revised, and take full responsibility for the final content, analytical design, tables, figures, and interpretations presented in the paper.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 265–283. https://doi.org/10.5555/3026877.3026899

Ansel, J., Yang, E., He, H., Gimelshein, N., Jain, A., Voznesensky, M., Bao, B., Bell, P., Berard, D., Burovski, E., Chauhan, G., Chourdia, A., Constable, W., Desmaison, A., DeVito, Z., Ellison, E., Feng, W., Gong, J., Gschwind, M., … & Chintala, S. (2024). PyTorch 2: Faster machine learning through dynamic Python bytecode transformation and graph compilation. In Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2024), Volume 2, pp. 929–947. https://doi.org/10.1145/3620665.3640366

Bai, J., Lu, F., Zhang, K., et al. (2019). ONNX: Open Neural Network Exchange. GitHub repository. https://doi.org/10.5281/zenodo.5541341

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13, 281–305. https://doi.org/10.5555/2188385.2188395

Björck, Å. (1996). Numerical methods for least squares problems. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611971484

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). JAX: Composable transformations of Python+NumPy programs. GitHub repository. https://doi.org/10.5281/zenodo.4724125

Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5–32. https://doi.org/10.1023/A:1010933404324

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., & Varoquaux,

G. (2013). API design for machine learning software: experiences from the scikit-learn project. arXiv preprint arXiv:1309.0238. https://doi.org/10.48550/arXiv.1309.0238

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), pp. 785–794. https://doi.org/10.1145/2939672.2939785

Coelho, C. N., Kuusela, A., Li, S., Zhuang, H., Ngadiuba, J., Aarrestad, T. K., Loncar, V., Pierini, M., Pol, A. A., & Summers, S. (2021). Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. Nature Machine Intelligence, 3(8), 675–686. https://doi.org/10.1038/s42256-021-00356-5

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In Advances in Neural Information Processing Systems, Vol. 35, pp. 16344–16359. https://doi.org/10.48550/arXiv.2205.14135

Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In Advances in Neural Information Processing Systems, Vol. 35, pp. 30318–30332. https://doi.org/10.48550/arXiv.2208.07339

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In Proceedings of the 9th International Conference on Learning Representations (ICLR 2021). https://doi.org/10.48550/arXiv.2010.11929

Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). GPTQ: Accurate post-training quantization for generative pre-trained transformers. In Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023). https://doi.org/10.48550/arXiv.2210.17323

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., & Keutzer, K. (2022). A survey of quantization methods for efficient neural network inference. In M. Thiruvathukal, Y. Lu, J. Kim, Y. Chen, & B. Chen (Eds.), Low-Power Computer Vision, pp. 291–326. Chapman and Hall/CRC. https://doi.org/10.1201/9781003162810-13

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? In Advances in Neural Information Processing Systems Datasets and Benchmarks Track, Vol. 35, pp. 507–520. https://doi.org/10.48550/arXiv.2207.08815

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hawks, B., Duarte, J., Fraser, N. J., Pappalardo, A., Tran, N., & Umuroglu, Y. (2021). Ps and Qs: Quantization-aware pruning for efficient low latency neural network inference. Frontiers in Artificial Intelligence, 4, 676564. https://doi.org/10.3389/frai.2021.676564

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), pp. 770–778. https://doi.org/10.1109/CVPR.2016.90

Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., & Pineau, J. (2020). Towards the systematic reporting of the energy and carbon footprints of machine learning. Journal of Machine Learning Research, 21(248), 1–43. https://doi.org/10.48550/arXiv.2002.05651

Horowitz, M. (2014). 1.1 Computing's energy problem (and what we can do about it). In 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), pp. 10–14. https://doi.org/10.1109/ISSCC.2014.6757323

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., … & Yoon, D. H. (2017). In-datacenter performance analysis of a Tensor Processing Unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA '17), pp. 1–12. https://doi.org/10.1145/3079856.3080246

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). https://doi.org/10.48550/arXiv.1412.6980

Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., & Han, S. (2024). AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. Proceedings of Machine Learning and Systems (MLSys 2024), 6, 87–100. https://doi.org/10.48550/arXiv.2306.00978

Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A ConvNet for the 2020s. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2022), pp. 11976–11986. https://doi.org/10.1109/CVPR52688.2022.01167

Lloyd, S. (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory, 28(2), 129–137. https://doi.org/10.1109/TIT.1982.1056489

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In Proceedings of the 7th International Conference on Learning Representations (ICLR 2019). https://doi.org/10.48550/arXiv.1711.05101

Mattson, P., Cheng, C., Diamos, G., Coleman, C., Micikevicius, P., Patterson, D., Tang, H., Wei, G.-Y., Bailis, P., Bittorf, V., Brooks, D., Chen, D., Dutta, D., Gupta, U., Hazelwood, K., Hock, A., Huang, X., Kang, D., Kanter, D., … & Zaharia, M. (2020). MLPerf training benchmark. Proceedings of Machine Learning and Systems (MLSys 2020), 2, 336–349. https://doi.org/10.48550/arXiv.1910.01500

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2017). Pointer sentinel mixture models. In Proceedings of the 5th International Conference on Learning Representations (ICLR 2017). https://doi.org/10.48550/arXiv.1609.07843

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., & Wu, H. (2018). Mixed precision training. In Proceedings of the 6th International Conference on Learning Representations (ICLR 2018). https://doi.org/10.48550/arXiv.1710.03740

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, Vol. 32, pp. 8026–8037. https://doi.org/10.48550/arXiv.1912.01703

Patterson, D., Gonzalez, J., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D., Texier, M., & Dean, J. (2021). Carbon emissions and large neural network training. arXiv preprint arXiv:2104.10350. https://doi.org/10.48550/arXiv.2104.10350

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., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://doi.org/10.5555/1953048.2078195

Rasley, J., Rajbhandari, S., Ruwase, O., & He, Y. (2020). DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '20), pp. 3505–3506. https://doi.org/10.1145/3394486.3406703

Reddi, V. J., Cheng, C., Kanter, D., Mattson, P., Schmuelling, G., Wu, C.-J., Anderson, B., Breughe, M., Charlebois, M., Chou, W., Chukka, R., Coleman, C., Davis, S., Deng, P., Diamos, G., Duke, J., Fick, D., Gardner, J. S., Hubara, I., … & Zhou, Y. (2020). MLPerf inference benchmark. In Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA 2020), pp. 446–459. https://doi.org/10.1109/ISCA45697.2020.00045

Reinhardt, A., Weber, L., & Gerstenberger, A. (2024). A decomposition of training FLOPs for transformer language models. Transactions on Machine Learning Research. https://doi.org/10.48550/arXiv.2403.02901

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. arXiv preprint arXiv:1910.01108. https://doi.org/10.48550/arXiv.1910.01108

Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. Information Fusion, 81, 84–90. https://doi.org/10.1016/j.inffus.2021.11.011

Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2020). Efficient processing of deep neural networks. Synthesis Lectures on Computer Architecture, 15(2), 1–341. https://doi.org/10.2200/S01004ED1V01Y202004CAC050

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. In Proceedings of the 36th International Conference on Machine Learning (ICML 2019), PMLR 97, pp. 6105–6114. https://doi.org/10.48550/arXiv.1905.11946

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society: Series B (Methodological), 58(1), 267–288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

Tornede, T., Tornede, A., Hanselle, J., Mohr, F., Wever, M., & Hüllermeier, E. (2023). Towards green automated machine learning: Status quo and future directions. Journal of Artificial Intelligence Research, 77, 427–457. https://doi.org/10.1613/jair.1.14340

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems, Vol. 30, pp. 5998–6008. https://doi.org/10.48550/arXiv.1706.03762

Wang, K., Liu, Z., Lin, Y., Lin, J., & Han, S. (2019). HAQ: Hardware-aware automated quantization with mixed precision. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2019), pp. 8604–8612. https://doi.org/10.1109/CVPR.2019.00881

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., & Rush, A. M. (2020). Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP 2020), pp. 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6