Can GPU Telemetry Verify LLM Training? Evidence from Temporal Structure in a Black-Box Setting

Aline Gabriel de Almeida [1]

Abstract

Many proposals for governing advanced AI systems rely on the ability to monitor or verify large-scale training runs, yet such verification often depends on access to logs, software instrumentation, or trusted reporting. This paper investigates whether temporal patterns in hardware-level GPU telemetry can provide workload-blind evidence for distinguishing LLM training from other computational workloads. We collect high-frequency telemetry traces in a controlled single-GPU setting across Llama and Qwen autoregressive LLM training and inference, BERT-base MLM training, and ResNet-50 CNN training. We analyze power signals using window-based features and a one-class verification approach. We find that autoregressive LLM training traces exhibit a distinctive temporal structure in GPU power telemetry in this setting. A verifier trained only on Llama-family training traces identifies held-out Qwen training runs as close to the reference distribution, while inference and ResNet-50 training are separated at larger distances. BERT-base remains closer to the boundary, suggesting graded similarity among transformer-based training workloads. These results show that hardware-originated telemetry can contain workload-relevant information even without access to model internals, logs, or datasets. They suggest a promising complementary pathway for verification in compute-based AI governance.

1. Introduction

Recent advances in artificial intelligence have been driven in large part by the rapid scaling of computational resources (Amodei & Hernandez, 2018). As a result, compute has emerged not only as a key input to capability development, but also as a promising lever for governance. Unlike data or algorithms, large-scale AI-relevant compute is physical, measurable, and concentrated within a relatively small number of actors, making it, in principle, amenable to monitoring and control (Sastry et al., 2024). A growing body of work has therefore proposed compute-based governance mechanisms, including reporting requirements, usage tracking, cloud-provider-mediated oversight, and thresholds on large-scale training runs (Sastry et al., 2024; Heim et al., 2024).

Central to these proposals is the assumption that the use of compute can be meaningfully observed. Effective governance depends not only on the ability to set rules, but also on the ability to verify compliance. In practice, this requires some form of visibility into how computational resources are being used (Sastry et al., 2024). Without such visibility, enforcement becomes difficult, and regulatory approaches relying on reporting or thresholds may be vulnerable to evasion or misreporting. This challenge is particularly salient in settings where access to workload internals, such as software stacks, datasets, training code, checkpoints, logs, or model weights is unavailable, limited, or strategically withheld.

A key difficulty arises from the fact that modern accelerators such as GPUs are general-purpose devices. The same hardware infrastructure can support a wide range of workloads, including large-scale language model training, inference, and other forms of machine learning or high-performance computing. As a result, observing that substantial compute is being used does not, by itself, reveal what type of computation is being performed. Existing approaches to monitoring often rely on internal metadata, such as workload logs, resource allocation statistics, or cloud-provider instrumentation (Seferis & Fist, 2026). While these signals can be informative, they may depend on trusted reporting, expose sensitive information, or fail in adversarial settings where workloads are deliberately structured or masked to evade detection (Seferis & Fist, 2026).

These limitations motivate the search for alternative signals that do not rely on privileged access to the workload itself. We study this question in a workload-blind, infrastructure-

[1]Independent Researcher, Brazil. Correspondence to: Aline Gabriel de Almeida <aline.gabriel.almeida@gmail.com>.

Second Workshop on Technical AI Governance Research (TAIGR) @ ICML 2026, Seoul, South Korea. 2026. Copyright 2026 by the author(s).

mediated setting, where the verifier observes telemetry exposed by the node or monitoring infrastructure but does not access the model, code, data, checkpoints, or workload-internal logs. Modern GPUs expose a range of low-level measurements, including power consumption, utilization, clock frequencies, and data movement, which are generated by on-board sensors and can be collected by the node or infrastructure monitoring layer without direct access to the underlying workload (Yang et al., 2024; NVIDIA Corporation, 2024; 2026a). Because these signals originate from the physical execution of computation, they may capture aspects of how computation unfolds over time, rather than just how much compute is used.

In this work, we investigate whether hardware-level telemetry signals preserve information about the temporal structure of different computational workloads. Specifically, we ask whether patterns in these signals can be used to distinguish between classes of workloads without relying on workload-internal information. We focus in particular on distinguishing autoregressive large language model (LLM) training from other workloads, including LLM inference and other neural-network training workloads.

To explore this question, we collect high-frequency telemetry traces from a single NVIDIA A100 GPU across multiple workloads, including LLM training, LLM inference, convolutional neural network (CNN) training, and masked language model (MLM) training. We analyze these signals using window-based features and distance-based comparisons, and evaluate whether they exhibit consistent, workload-specific structure.

Our results provide initial evidence that different workloads induce distinct temporal patterns in hardware-level telemetry signals. These patterns are visible in raw traces, persist in feature representations, and lead to high empirical separability in this controlled setting under simple statistical methods. Notably, a verifier fitted on Llama-family training traces also identifies Qwen training runs as close to the training distribution, suggesting a non-trivial generalization signal across model families in this setting. While these findings are preliminary and limited to a controlled experimental setting, they suggest that non-intrusive, hardware-originated signals may contain meaningful information about how computation is organized over time.

More broadly, this work highlights a potential direction for supporting compute-based governance, in which existing approaches based on reporting and metadata are complemented by signals derived from physical measurements. If such signals can be shown to generalize and remain robust under realistic conditions, they could contribute to new forms of black-box verification that do not rely on access to sensitive workload-internal information.

2. Problem Setting

Recent proposals for compute governance introduce regulatory mechanisms that explicitly target large-scale AI training runs. For example, policy frameworks have proposed reporting requirements for training runs above specified compute thresholds, licensing regimes tied to frontier model development, and monitoring systems implemented at the level of cloud or hardware providers (Heim et al., 2024; Sastry et al., 2024). These proposals do not aim to regulate compute usage in general, but specifically to identify and constrain training processes that may produce advanced capabilities.

This focus on training is not incidental. Training is the phase in which model parameters are created and capabilities are acquired, making it a central point of intervention in governance frameworks concerned with advanced AI development (Amodei & Hernandez, 2018). In contrast, inference and other uses of compute typically apply existing capabilities rather than creating new ones, and are therefore often treated as lower-risk in compute governance frameworks.

Together, these proposals create a concrete technical requirement for monitoring systems to distinguish between different classes of workloads executed on general-purpose hardware. In particular, they must differentiate large-scale training runs from other workloads, including inference and non-ML computation.

Existing approaches address this requirement under different assumptions about system access. Some approaches rely on infrastructure-level metadata. For example, proposals in Heim et al. (2024) and Sastry et al. (2024) assume access to cloud-provider metadata, including logs, job scheduling information, and allocation records, to track and audit large-scale training runs. These approaches can provide detailed visibility into workload execution, but depend on access to provider-controlled systems and on accurate workload classification or reporting. Beyond such metadata, managed GPU infrastructure can also expose operational telemetry from the hardware itself. Recent operational analysis of LLM pre-training clusters uses Prometheus and GPU telemetry to diagnose failures and performance bottlenecks at multi-node scale (Lablup Inc., 2026). This illustrates that hardware telemetry is practically observable in managed infrastructure, although it is typically used for operations rather than workload verification.

A different source of visibility comes from software-level instrumentation. Profiling tools and training-integrated logging expose fine-grained information about model execution, including iteration structure and resource usage. However, these methods require direct integration with the workload itself. This makes them especially informative when available, but less applicable when the workload owner is unable or unwilling to expose internal execution details.

Closer to the present work, some approaches explicitly study workload classification using system-level signals. Seferis & Fist (2026) discuss detection approaches for compute structuring, where actors may split, disguise, or mask training workloads to avoid compute-governance thresholds. In their framing, workload classification is part of a broader monitoring problem. The approaches they discuss combine hardware metrics such as power, utilization, and bandwidth with contextual information about execution environments, provider metadata, or aggregation across nodes. This additional context can help disambiguate workloads that may otherwise appear similar under coarse hardware measurements. By contrast, the present work asks whether temporal structure in a single hardware telemetry signal can itself carry workload-relevant information when workload-internal logs, model details, and contextual metadata are absent.

A related body of work studies hardware-level observation more directly. Prior GPU side-channel work has shown that memory behavior, timing, and performance counters can reveal information about co-located workloads, including neural-network execution (Naghibijouybari et al., 2021). More recent work has applied hardware-level measurement to AI oversight and governance. Xu et al. (2026) use GPU performance monitoring counters to verify properties of LLM inference without directly inspecting model weights or user data, while Monfared et al. (2026) study timing and memory telemetry mechanisms for GPU governance under weak-trust assumptions. These works show that hardware-originated signals can carry governance-relevant information, but they differ from the setting studied here in their objectives and measurement assumptions. Prior work has focused on side-channel leakage, LLM-inference oversight, or active timing and memory probes, whereas this work asks whether passively observed power telemetry exposed by the node or monitoring infrastructure contains temporal structure that can help distinguish training-like workloads from other workloads.

Despite their differences, existing approaches generally require some form of information beyond power telemetry alone, such as richer hardware counters, active probing, co-location assumptions, or contextual system information. Infrastructure-based methods require access to cloud metadata, instrumentation-based methods require integration with the workload, and system-level classification approaches depend on contextual information beyond raw hardware signals. In settings where monitoring occurs across organizational boundaries, or where actors may have incentives to obscure workload characteristics, these assumptions may not hold (Seferis & Fist, 2026).

This motivates studying workload-blind hardware telemetry under more limited observation, where relying solely on hardware-level telemetry introduces a different challenge. Aggregate metrics such as average power consumption or utilization do not uniquely identify the underlying computation, as different workloads can produce similar coarse-grained resource usage profiles. Prior work has highlighted the difficulty of inferring fine-grained workload properties from such signals, particularly when measurements are low-frequency or aggregated (Yang et al., 2024).

These concerns primarily apply to aggregate statistics, leaving open whether richer temporal structure in the signals may carry additional information about how computation unfolds over time.

In this work, we consider a workload-blind setting in which only hardware-level telemetry signals are available, collected from on-board GPU sensors through the node or monitoring infrastructure. We focus primarily on power consumption as a representative signal, while noting that similar measurements (e.g., utilization or clock frequencies) are often available in practice. No access is assumed to logs, model architectures, datasets, checkpoints, training code, or orchestration metadata. Formally, we observe a time series

$$x(t), \quad t = 1, \ldots, T,$$

and seek to infer properties of the underlying workload from the observed telemetry signal. In particular, we study the task of distinguishing LLM-training-like workloads from other workloads, including LLM inference and other forms of neural-network training.

This formulation departs from prior work by removing access to the contextual and system-level information that existing approaches rely on, and instead asks whether workload-level inference is possible from raw physical signals alone. Rather than relying on aggregate statistics, we focus on temporal structure, motivated by the hypothesis that while different workloads may exhibit similar average resource usage, they may differ in how computation evolves over time.

The resulting problem is inherently challenging. Machine learning workloads share common structural patterns, such as iterative updates and batched execution, and telemetry signals are indirect and subject to sampling limitations (Yang et al., 2024). Any discriminative information must therefore arise from higher-order temporal properties rather than direct indicators of workload type.

Our central hypothesis is that such temporal structure exists and is preserved in hardware-level telemetry signals. If so, it may enable workload-level inference in a workload-blind, infrastructure-mediated setting, without relying on workload-internal access or trusted reporting. The remainder of this work evaluates this hypothesis empirically.

3. Experimental Setup

We investigate whether hardware-level telemetry encodes consistent and workload-specific structure across different computational processes by collecting high-frequency telemetry traces under controlled conditions.

3.1. Workloads

We evaluate four classes of workloads: autoregressive LLM training, LLM inference, MLM training, and CNN training. Autoregressive LLM workloads include both Llama 3.2 3B (Dubey et al., 2024) and Qwen 2.5 3B (Yang et al., 2025) models. MLM training is performed using BERT-base (Devlin et al., 2019), and CNN training uses ResNet-50 (He et al., 2016).

Each workload is executed under multiple configurations to capture variability within each workload class.

These configurations vary key parameters such as batch size, sequence length, number of generated tokens, or input resolution. The variations induce differences in the temporal structure of computation within each workload class. The parameter ranges are summarized in Table 1.

Table 1. Workload configurations used for telemetry collection. Each workload is executed under multiple parameter settings to capture variability within each workload class. B denotes batch size, GA gradient accumulation, T the maximum number of generated tokens, L sequence length, and R input resolution.

Workload	Model	Parameter ranges
LLM (train)	Llama 3.2 3B	$(B, GA) \in \{1, 2\} \times \{1, 2\}$
	Qwen 2.5 3B	$(B, GA) \in \{1, 2\} \times \{1, 2\}$
LLM (inf)	Llama 3.2 3B	$(B, T) \in \{32, 64\} \times \{32, 512\}$
	Qwen 2.5 3B	$(B, T) \in \{32, 64\} \times \{32, 512\}$
MLM (train)	BERT-base	$(B, L) \in \{20, 40\} \times \{128, 512\}$
CNN (train)	ResNet-50	$(B, R) \in \{256, 512\} \times \{160, 224\}$

3.2. Data Collection

All experiments are conducted on a single NVIDIA A100-SXM4-40GB GPU. Each workload configuration is executed once, for 30 minutes, providing steady-state segments for window-based analysis.

Telemetry signals are collected using a custom-built sampler based on NVIDIA's NVML interface (NVIDIA Corporation, 2024). The sampler is implemented in C and operates independently of the workload process, allowing telemetry collection without modifying training or inference code. The design is inspired by prior work on GPU power measurement (Guerreiro et al., 2018), but reimplemented and adapted for high-frequency, continuous telemetry collection in a black-box setting.

This sampler serves as the experimental collection mechanism. In infrastructure settings, analogous telemetry could be collected through provider-operated GPU monitoring systems such as NVIDIA Data Center GPU Manager (DCGM) or DCGM-Exporter, through cloud telemetry pipelines, or through governance-specific monitoring layers designed to expose comparable hardware measurements, without modifying the workload code or accessing model internals (NVIDIA Corporation, 2026a;b).

Sampling is performed at a target interval of 10 ms. In practice, this yields an effective sampling rate of approximately 30–40 Hz due to system-level overhead and operating system scheduling. Sampling jitter and effective rate variability are accounted for in downstream analysis by using window-based aggregation.

To reduce measurement interference, telemetry collection is isolated from the workload at the CPU level. The sampler process is pinned to a dedicated CPU core, while the workload is restricted to the remaining cores. This prevents contention between telemetry collection and host-side workload activity (e.g., data loading and runtime scheduling), improving measurement stability and reducing sampling-induced artifacts. Details of the execution setup, including CPU affinity and process isolation, are provided in Appendix A.

All analyses are performed on steady-state segments of each run, excluding initialization phases where behavior is dominated by transient system effects rather than sustained computation.

4. Verification Method

We formulate workload identification as a one-class verification problem relative to a reference distribution constructed from autoregressive LLM training runs, asking whether a given telemetry trace is statistically consistent with the empirical temporal patterns observed in the reference training runs.

4.1. Window-Level Representation

Let $x(t)$ denote the run-normalized GPU power trace extracted from the steady-state portion of a run. We partition each trace into non-overlapping windows of fixed duration $\Delta \in \{5\text{s}, 10\text{s}\}$, producing a sequence of windows

$$w_1, w_2, \ldots, w_N.$$

For each window, we compute a fixed-dimensional feature vector

$$\mathbf{z}_i \in \mathbb{R}^{12},$$

using twelve summary features of the run-normalized power signal. These features are grouped into three families. The first captures variability, using interquartile range, median absolute deviation, and line length. The second captures temporal structure, using autocorrelation at lags 1 and 5, autocorrelation energy up to lag 10, and centered zero-crossing rate. The third captures spectral shape, using spectral entropy, peak power ratio, and low-, mid-, and high-band power ratios computed relative to the Nyquist frequency. This transforms each run from a raw time series into a sequence of window-level feature representations.

Although other telemetry channels were recorded, the verification experiments in this paper use only the GPU power signal as an initial single-channel analysis.

4.2. Reference Model for LLM Training

We construct the reference model using only autoregressive LLM training windows from the Llama family. Let

$$\mathcal{Z}_{\text{train}} = \{\mathbf{z}_i^{(\text{Llama train})}\}$$

denote the set of training windows at a fixed temporal scale.

We first standardize features using the mean and standard deviation estimated from $\mathcal{Z}_{\text{train}}$, and then fit a diagonal Gaussian model in the standardized feature space. This yields a reference distribution with mean vector $\boldsymbol{\mu}$ and diagonal covariance matrix Σ. The diagonal covariance assumption provides a simple baseline model and avoids estimating a full covariance matrix from a limited number of reference runs.

This model defines an empirical reference distribution for Llama-family LLM training at that temporal scale.

4.3. Window-Level Distance

Given a window-level feature vector $\mathbf{z}$, after applying the same standardization used to fit the reference model, we obtain $\tilde{\mathbf{z}}$ and compute its Mahalanobis distance to the LLM training reference distribution:

$$d(\tilde{\mathbf{z}}) = \sqrt{(\tilde{\mathbf{z}} - \boldsymbol{\mu})^\top \Sigma^{-1} (\tilde{\mathbf{z}} - \boldsymbol{\mu})}.$$

Windows with smaller distances are more consistent with the learned reference distribution, while larger distances indicate deviations from that empirical training reference.

4.4. Run-Level Aggregation

Applying this procedure to all windows in a run yields a sequence of distances

$$d_1, d_2, \ldots, d_N.$$

We aggregate these to the run level using summary statistics including mean distance, median distance, and the fraction of windows whose distances fall below a verification threshold. These aggregated scores capture, respectively, overall closeness to the LLM-training reference distribution, robustness to outlier windows, and the proportion of windows classified as training-consistent.

4.5. Decision Rule

We define the verification threshold using the distribution of Mahalanobis distances over the LLM training windows used to fit the reference model. Specifically, the threshold is set to the 95th percentile of the training-window distance distribution. The 95th percentile is used here as a fixed operating point for this proof-of-concept evaluation, with threshold calibration left to deployment-specific settings.

A window is classified as consistent with LLM training if its distance falls below this threshold. Run-level decisions are then derived from the aggregated scores, including majority-style criteria based on the proportion of below-threshold windows.

4.6. Evaluation

We evaluate the method at two temporal resolutions, using 5-second and 10-second windows. The reference model is fit using only Llama-family autoregressive LLM training windows, and evaluation is performed on all workloads, including held-out Qwen LLM training, LLM inference, MLM training, and CNN training.

Performance is summarized using run-level metrics derived from the aggregated distance scores, including AUC, majority-vote accuracy, and false positive and false negative rates. No threshold tuning is performed on the evaluation workloads.

5. Results

We evaluate whether the proposed verification pipeline can distinguish LLM training from other workloads using only black-box, workload-blind power telemetry. Unless otherwise noted, all analyses are performed on steady-state segments (200 s–1800 s), and the reference model is fit using windows from Llama-family autoregressive training runs.

5.1. Power traces show distinct temporal structure across workloads

We first inspect short steady-state segments of the raw power signal to establish what is already visible in black-box workload-blind telemetry before any feature extraction.

Figure 1 compares LLM training and inference for two

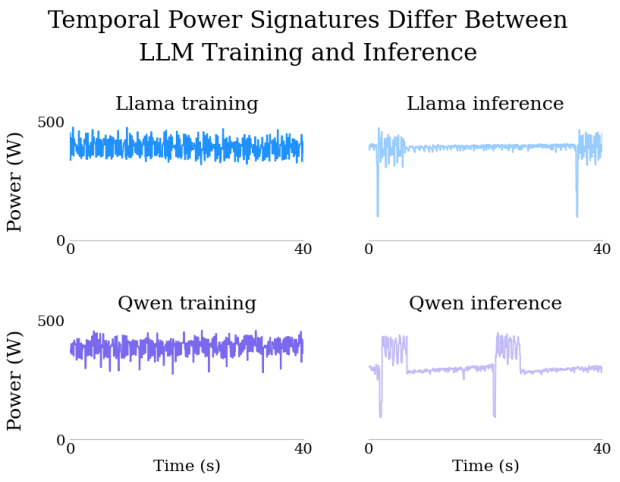

Figure 1. Representative 40 s steady-state power traces for LLM training and inference on an NVIDIA A100 GPU. Across both model families, the training traces exhibit sustained high-power behavior with relatively continuous fluctuations, whereas the inference traces show more segmented temporal dynamics, including flatter plateaus, abrupt transitions, and transient drops.

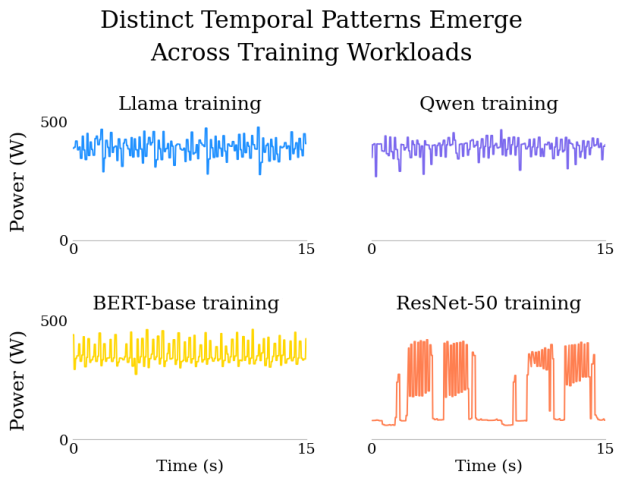

Figure 2. Representative 15 s steady-state power traces across training workloads. Autoregressive LLM training remains relatively continuous over time, BERT-base training shows stronger periodic structure, and ResNet-50 training exhibits pronounced alternation between low-power intervals and bursty high-power phases.

model families, Llama and Qwen. In both families, the training traces exhibit sustained high-power behavior with dense, continuously varying fluctuations. The inference traces show a different temporal pattern, with a more segmented signal, flatter intervals, sharper transitions, and transient drops that are less prominent in training. The contrast is visible in both model families, suggesting that the difference is not specific to a single model.

Figure 2 broadens the comparison to different training workloads. Llama and Qwen training remain visually similar, with dense and continuously varying high-power traces. BERT-base training is also sustained, but shows a more regular internal rhythm, with stronger visible periodicity. ResNet-50 training is more clearly separated, alternating between lower-power intervals and bursty high-power phases and producing a markedly different temporal pattern from the transformer-based workloads.

Two points are notable. First, sustained high power alone does not imply a common temporal signature, since even among training workloads, the traces differ substantially in how computation unfolds over time. Second, similarity is graded rather than binary. BERT-base appears closer to LLM training than ResNet-50 does, which is plausible given that BERT and LLMs are both transformer-based text workloads. At the same time, BERT does not collapse onto the autoregressive LLM-training pattern, as its traces remain more periodic and less continuously varying.

Taken together, Figures 1 and 2 indicate that raw power telemetry already carries visible workload-dependent temporal structure. We next evaluate whether the verification pipeline preserves this structure for quantitative separation.

5.2. Reference-based distance separates autoregressive LLM training from other workloads

We evaluate run-level verification by measuring distance to an autoregressive LLM training reference.

A diagonal Gaussian reference model is fit using Llama training windows only. Figure 3 shows the resulting run-level mean Mahalanobis distances for 5 s windows, where smaller values indicate greater consistency with the learned reference distribution. The dashed line marks the decision threshold defined by the 95th percentile of training-window distances.

Several patterns are apparent. The Llama training runs used to fit the reference form a compact low-distance cluster. Qwen training runs, although not used during fitting, remain close to this region and below the threshold, indicating that the learned representation captures structure that transfers from Llama-family training to Qwen training in this setting, rather than memorizing a specific model.

Inference runs from both Llama and Qwen are consistently shifted upward and lie above the threshold, separating them from training despite sharing the same architectures. This indicates that the reference is sensitive to the distinction between training and inference within the tested model fam-

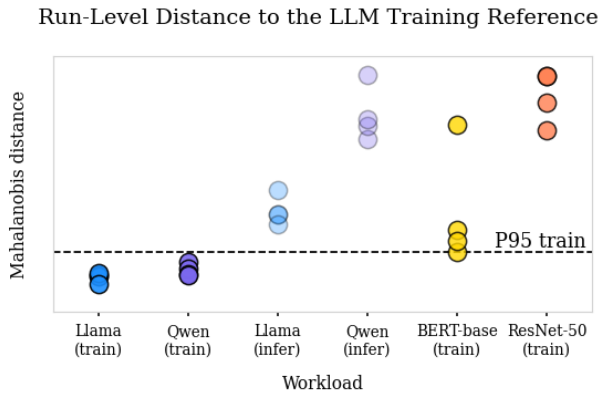

Figure 3. Run-level mean Mahalanobis distance to the Llama-family LLM-training reference for 5 s windows. The dashed line marks the 95th-percentile training-window threshold. Qwen training, unseen during fitting, remains close to the reference, while inference and ResNet-50 are separated at larger distances. BERT-base lies near the threshold.

ilies, rather than simply to model family.

Other training workloads exhibit distinct behavior. BERT-base lies near the decision boundary, with distances overlapping the threshold region, whereas ResNet-50 is clearly separated at substantially larger distances. This indicates that similarity to the reference is graded, with transformer-based training on text remaining partially aligned with the autoregressive LLM-training reference, consistent with shared computational structure and data modality, while CNN-based training on images follows a more distinct execution pattern.

Table 2 quantifies this separation at the run level for both 5 s and 10 s windows. Across both temporal scales, the run-level metrics indicate high separability in this controlled dataset. Mean-distance AUC reaches 0.984 for 5 s windows and 1.000 for 10 s windows, while median-distance AUC is 0.992 at both scales. Majority-vote accuracy is 0.917 at 5 s and 0.958 at 10 s. Given the limited number of runs, these metrics should be interpreted descriptively.

Table 2. Run-level verification performance relative to the Llama-family LLM-training reference. Metrics are computed from aggregated Mahalanobis-distance scores. Threshold-based metrics use the 95th percentile of training-window distances.

Metric	5 s windows	10 s windows
AUC (mean distance)	0.984	1.000
AUC (median distance)	0.992	0.992
Accuracy (majority vote)	0.917	0.958
False positive rate	0.083	0.000
False negative rate	0.000	0.042

The comparison across window lengths reveals a trade-off in this dataset. The 10 s setting yields cleaner aggregate separation and eliminates false positives under the threshold rule in this experiment. The 5 s setting, however, avoids false negatives, preserving sensitivity to LLM training behavior. This indicates that temporal resolution can shape the balance between specificity and sensitivity in verification.

6. Discussion

6.1. Scope and Limitations

This work provides a proof of concept that black-box, workload-blind hardware telemetry can retain structured information about how computation unfolds over time, and that this information can be used to distinguish autoregressive LLM training from other workloads under controlled conditions.

The results should not be interpreted as a general-purpose detection system. The experiments are limited in scope, and the observed separability is best understood as evidence of signal presence rather than as a deployment-ready capability. In particular, the current setup does not evaluate robustness across variation in hardware platforms, software stacks, training configurations, distributed-training regimes, precision formats, compiler settings, power-management policies, or execution environments.

Several factors may influence the stability of the observed patterns. Differences in model implementation, data pipelines, scheduling, and system-level orchestration could alter the temporal structure captured by the telemetry. In addition, the experiments are conducted in a single-node, single-GPU setting, and do not address distributed training, where gradient synchronization, communication overhead, pipeline parallelism, and sharding strategies may introduce additional sources of variability. These dynamics may complicate direct transfer from single-GPU traces, but they may also introduce additional temporal structure that could make large-scale training more distinguishable rather than less.

The central contribution is therefore more specific. This work shows that temporal structure in physical GPU signals is not fully abstracted away at the hardware level, and that this structure can be recovered without access to model internals, logs, or datasets. This provides initial evidence for workload-blind, infrastructure-mediated verification based on hardware telemetry, while leaving open the question of how robust such signals remain under more realistic conditions.

6.2. Implications for Compute Governance

Many proposals for compute-based governance rely on the ability to monitor or verify large-scale training activity. In

practice, such approaches often depend on access to internal logs, software instrumentation, or trusted reporting, which may not be available or may be strategically withheld.

The results in this work suggest a complementary direction. If hardware-level telemetry retains information about how computation unfolds over time, then hardware-originated telemetry signals may provide partial and indirect visibility into the nature of the workload being executed, without requiring access to model internals or training data. This is particularly relevant in settings where direct inspection is infeasible or undesirable.

Importantly, the findings do not imply that LLM training can be reliably detected in realistic environments using the present method. Rather, they indicate that the underlying signals required for such detection may exist. This shifts part of the verification problem from one of signal availability to one of robustness, modeling, and deployment. Deployment would also require accounting for asymmetric error costs. In governance settings, false negatives may be especially costly, since missed detections could allow the most relevant training runs to proceed unnoticed. Threshold selection would therefore need to reflect the consequences of missed detection, not only aggregate classification performance.

6.3. Open Questions

The results raise several questions that are central to assessing whether this approach can support verification in practice.

First, how stable are the observed patterns across changes in hardware and software? Differences in GPU architecture, driver behavior, compiler settings, and training frameworks may affect both the form and consistency of the telemetry signal.

Second, how does the signal behave under variation in training configuration? Changes in batch size, sequence length, optimization strategy, precision regime, and data pipeline may alter the temporal structure in ways that affect separability.

Third, how does the approach extend to distributed training? Large-scale LLM training is typically performed across many devices, where synchronization, communication, sharding, and pipeline parallelism introduce additional dynamics that are not captured in a single-GPU setting.

Fourth, how distinguishable are closely related workloads? While the present results separate LLM training from inference and from CNN training, BERT-base lies near the decision boundary, and it remains unclear how well different LLM training regimes, architectures, or optimization strategies can be distinguished from one another.

Finally, how robust are these signals under adversarial conditions? If telemetry-based verification becomes relevant for governance, actors may have incentives to modify execution patterns to obscure or mimic reference patterns. A central governance question is how costly such mimicry would be. Evasion strategies such as power capping, frequency locking, compiler changes, or dummy-compute injection may make the signal less reliable, but they may also require wasted computation, degraded performance, or constrained system operation. If so, the signal could still have enforcement value even when it is not perfectly robust.

7. Conclusion

This work investigated whether workload-blind GPU power telemetry retains information about how computation unfolds over time, and whether such information can support LLM-training verification.

Across a controlled set of workloads, we find that simple features extracted from power traces are sufficient to distinguish autoregressive LLM training from inference and CNN training, while BERT-base remains a closer boundary case. A one-class verification approach, trained only on Llama-family training data, recovers a structured ordering of workloads and provides descriptive evidence of run-level separability.

These results should be interpreted as evidence of signal, not as a complete detection system. They show that temporal structure in hardware-level signals is not fully abstracted away, and that this structure can be recovered without access to model internals, logs, or datasets.

More broadly, the findings suggest that hardware-originated telemetry signals may contain information relevant for verification. This points to a potential pathway for compute-based governance that does not rely exclusively on internal reporting or software-level instrumentation, while leaving open the challenge of achieving robustness in realistic and adversarial settings.

Impact Statement

This work investigates whether hardware-level telemetry can retain information about how computation unfolds over time, and whether such signals can support the workload-blind verification of LLM-training-like workloads without access to model internals.

If robust, this approach could contribute to the development of non-intrusive monitoring mechanisms for large-scale AI training, which has been proposed as a component of compute governance frameworks. In contrast to approaches that rely on internal logs or software instrumentation, hardware-originated signals may offer a form of infrastructure-mediated visibility that does not depend on

access to proprietary models, datasets, or workload-internal information.

At the same time, if telemetry-based verification is used for governance, it may create incentives to modify or obscure observable patterns. This introduces a potential dynamic in which monitoring approaches shape the behavior they aim to detect, raising questions about robustness under adaptive or adversarial conditions.

This work does not provide a deployable monitoring system, nor does it evaluate these broader implications in realistic settings. Rather, it shows that such signals may contain structured information. This shifts part of the challenge from whether hardware-originated telemetry signals exist to whether they can be made reliable, robust, and appropriately governed in practice.

LLM Usage Statement

LLMs were used to assist with literature exploration, English-language editing, and revision of the manuscript for clarity, flow, redundancy, and consistency of claims. All suggestions were reviewed and edited by the author. The author verified the relevant literature and remains responsible for the paper's content, analysis, and conclusions.

References

Amodei, D. and Hernandez, D. Ai and compute. https://openai.com/research/ai-and-compute, 2018. OpenAI Blog.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423/.

Dubey, A. et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Guerreiro, J., Ilic, A., Roma, N., and Tomas, P. Gpgpu power modeling for multi-domain voltage-frequency scaling. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 789–800, 2018. doi: 10.1109/HPCA.2018.00072.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

Heim, L., Fist, T., Egan, J., Huang, S., Zekany, S., Trager, R., Osborne, M. A., and Zilberman, N. Governing through the cloud: The intermediary role of compute providers in ai regulation, 2024. URL https://arxiv.org/abs/2403.08501.

Lablup Inc. From detection to recovery: Operational analysis on LLM pre-training with 504 GPUs, 2026. URL https://arxiv.org/abs/2605.09370. Technical report.

Monfared, S. K., Ganji, F., Holcomb, D., and Tajik, S. Timing and memory telemetry on gpus for ai governance, 2026. URL https://arxiv.org/abs/2602.09369.

Naghibijouybari, H., Neupane, A., Qian, Z., and Abu-Ghazaleh, N. Side channel attacks on gpus. *IEEE Transactions on Dependable and Secure Computing*, 18(4): 1950–1961, 2021. doi: 10.1109/TDSC.2019.2944624.

NVIDIA Corporation. Nvidia management library (nvml). https://developer.nvidia.com/nvidia-management-library-nvml, 2024. Accessed: 2026-04.

NVIDIA Corporation. NVIDIA Data Center GPU Manager (DCGM). https://developer.nvidia.com/dcgm, 2026a. Accessed: 2026-06.

NVIDIA Corporation. DCGM-Exporter: NVIDIA GPU Metrics Exporter for Prometheus. https://github.com/NVIDIA/dcgm-exporter, 2026b. Accessed: 2026-06.

Sastry, G., Heim, L., Belfield, H., Anderljung, M., Brundage, M., Hazell, J., O'Keefe, C., Hadfield, G. K., Ngo, R., Pilz, K., Gor, G., Bluemke, E., Shoker, S., Egan, J., Trager, R. F., Avin, S., Weller, A., Bengio, Y., and Coyle, D. Computing power and the governance of artificial intelligence, 2024. URL https://arxiv.org/abs/2402.08797.

Seferis, E. and Fist, T. Detecting compute structuring in ai governance is likely feasible. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(44): 37904–37912, Mar. 2026. doi: 10.1609/aaai.v40i44.41127. URL https://ojs.aaai.org/index.php/AAAI/article/view/41127.

Xu, H., Gong, C., Liu, B., Zheng, H., Chen, B., and Li, M. WAVE: Leveraging architecture observation for privacy-preserving model oversight. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, pp. 2212–2231, New York,

NY, USA, 2026. Association for Computing Machinery. doi: 10.1145/3779212.3790247. URL https://doi.org/10.1145/3779212.3790247.

Yang, A. et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Yang, Z., Adamek, K., and Armour, W. Accurate and convenient energy measurements for gpus: A detailed study of nvidia gpu's built-in power sensor. In *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–17. IEEE, November 2024. doi: 10.1109/sc41406.2024.00028. URL http://dx.doi.org/10.1109/SC41406.2024.00028.

Appendix

A. Measurement Protocol

All experiments were conducted on a single-node system equipped with an NVIDIA A100-SXM4-40GB GPU. The host machine runs Ubuntu 22.04 on an AMD EPYC processor with 216GB of RAM, under a virtualized environment.

Telemetry was collected via a standalone process using NVIDIA's NVML interface at a target interval of approximately 10 ms (effective sampling rate $\sim$30 – 40 Hz). We record *instantaneous* GPU power draw (NVML `power.draw`), rather than the 1-second averaged power metric, to preserve high-frequency temporal structure. According to NVIDIA documentation, instantaneous power readings may exhibit measurement noise on the order of $\sim$5 W; we do not smooth this signal.

To reduce measurement interference, telemetry collection is isolated from the workload at the CPU level. The sampler process is pinned to a dedicated CPU core, while the workload is restricted to a disjoint set of cores for all runs, ensuring strict separation between telemetry collection and host-side workload execution.

This prevents contention between telemetry collection and host-side activity (e.g., data loading and runtime scheduling).

All workloads operate on datasets that are pre-loaded and preprocessed prior to measurement. This avoids I/O-induced variability and ensures sustained GPU utilization during execution.

Each run consists of a single workload executed in isolation, with continuous telemetry recording and no concurrent GPU processes. Analysis is restricted to steady-state segments, excluding an initial warm-up period (first 200 s) to remove initialization effects such as model loading, compilation, and data pipeline ramp-up.

Telemetry includes synchronized time series of GPU power draw, utilization, clock frequencies, PCIe throughput, and temperature. The verification experiments in the main text use only the GPU power signal. Measurements are collected without modifying the workload process or accessing model internals, code, data, or logs. The setting therefore assumes access to hardware telemetry exposed by the node or infrastructure layer, rather than access to workload-internal information.