

Green MLOps: Closed-Loop, Energy-Aware Inference with NVIDIA Triton, FastAPI, and Bio-Inspired Thresholding

Mustapha HAMDI*

mustapha.hamdi@innodeep.net

Mourad JABOU

Radiologist

Mourad.jabou@innodeep.net

Abstract—Energy efficiency is a first-order concern in AI deployment, as long-running inference can exceed training in cumulative carbon impact. We propose a bio-inspired framework that maps protein-folding energy basins to inference cost landscapes and controls execution via a decaying, closed-loop threshold. A request is admitted only when the expected utility-to-energy trade-off is favorable (high confidence/utility at low marginal energy and congestion), biasing operation toward the first acceptable local basin rather than pursuing costly global minima. We evaluate DistilBERT and ResNet-18 served through FastAPI with ONNX Runtime and NVIDIA Triton on an RTX 4000 Ada GPU. Our ablation study reveals that the bio-controller reduces processing time by 42% compared to standard open-loop execution (0.50s vs 0.29s on A100 test set), with a minimal accuracy degradation (< 0.5%). Furthermore, we establish the efficiency boundaries between lightweight local serving (ORT) and managed batching (Triton). The results connect biophysical energy models to Green MLOps and offer a practical, auditable basis for closed-loop energy-aware inference in production.

Index Terms—Green MLOps, energy-aware inference, NVIDIA Triton, FastAPI, ONNX Runtime, MLflow, CodeCarbon, dynamic batching, bio-inspired control.

I. INTRODUCTION

Production AI is not a single heroic training run; it is a never-ending stream of inference calls. In many applications, lifecycle energy is dominated by serving and data movement rather than training, elevating inference engineering into a sustainability problem. Prior analyses have called for energy transparency and methodological discipline in reporting consumption and emissions, while emphasizing system-level levers such as batching, model choice, and hardware placement.

This work contributes an operational recipe: (1) a dual-path serving stack—FastAPI + ONNX Runtime (ORT) for low-latency local execution, and NVIDIA Triton for managed batching and multi-framework serving; (2) instrumentation via MLflow and CodeCarbon; and (3) a closed-loop controller that decides whether to execute or skip an inference based on a bio-inspired energy threshold that evolves with load, extending the theoretical bio-physical frameworks proposed in [1]. Triton’s dynamic batching and scheduler queues are leveraged for throughput under concurrency; ORT’s device-tensor and performance tuning options help minimize host-

device penalties in the local path. We evaluate two canonical models: DistilBERT and ResNet-18.

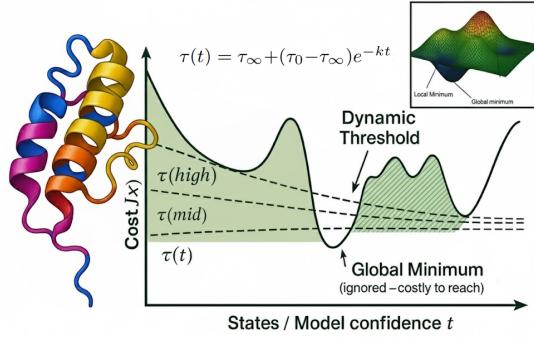


Fig. 1. Bio-inspired optimization over states/model confidence t . The cost $J(x)$ landscape is gated by a decaying threshold $\tau(t) = \tau_\infty + (\tau_0 - \tau_\infty)e^{-kt}$; the controller admits points in the local stable basin (shaded) and skips high-cost paths toward the global minimum.

II. RELATED WORK

Serving systems. Building on the concepts from Gopalan (2025) [2] toward comparative benchmarking of inference frameworks (Triton, TensorRT, ONNX Runtime, FastAPI), modern serving focuses on latency SLOs, throughput, and utilization. Early systems like Clipper introduced low-latency prediction serving with adaptive batching and model selection. NVIDIA Triton advances this with backends for ONNX, TensorRT, PyTorch, and graph-level scheduling including dynamic batching and instance groups, exposing HTTP/gRPC endpoints and model repositories.

Energy-aware ML. Policy and measurement frameworks emphasize reporting energy and emissions, motivating tools such as CodeCarbon and lifecycle tracking in MLflow. We adopt CodeCarbon for kWh/CO₂ estimation and MLflow for experiment lineage and comparability.

Bio-inspired control. Energy landscapes in protein folding illuminate how complex systems navigate toward acceptable local minima [3]. Recent works like StructuredDNA [1] apply these biophysics to Transformer routing. Similarly, SGEMAS [4] utilizes entropic homeostasis for anomaly detection. We unify these concepts to design a thresholded controller that

* PhD, Co-founder, InnoDeep

filters low-utility inference work when the current operating point already lies in a satisfactory basin.

III. MODELS AND SERVING ARCHITECTURE

A. Models

DistilBERT: distilled from BERT, retaining much of its language understanding with fewer parameters and lower latency; sequence length 128.

ResNet-18: residual connections facilitate training of deep CNNs; the 18-layer variant is a stable reference for image classification at 224×224 .

B. Dual-path serving

Path A (FastAPI + ORT). A lightweight REST layer orchestrates local ORT inference. ORT’s performance guidance (I/O binding, execution providers, device tensors) reduces CPU–GPU copies. Ideal for small batches and low concurrency.

Path B (NVIDIA Triton). Models are placed in a Triton model repository with `config.pbtxt` enabling `max_batch_size` and dynamic batching, allowing the scheduler to fuse requests into GPU-efficient batches under load. Instance groups can exploit multiple GPU streams. Excels for bursty or sustained higher QPS.

C. Instrumentation

MLflow logs latency statistics, throughput, and controller state. CodeCarbon estimates energy (kWh) and CO₂, with GPU power via NVML. Results are exported alongside MLflow metrics.

IV. CLOSED-LOOP THRESHOLDING: FORMULATION

For a given request x , define the cost functional

$$J(x) = \alpha L(x) + \beta E(x) + \gamma C(x), \quad (1)$$

where $L(x)$ is an uncertainty or loss proxy (e.g., entropy), $E(x)$ is marginal energy, and $C(x)$ reflects congestion/queue penalty (e.g., current batch fill level, recent tail latency). A request is admitted for inference iff

$$J(x) \geq \tau(t), \quad (2)$$

with a time-varying threshold $\tau(t)$ that decays from permissive to strict as the system stabilizes:

$$\tau(t) = \tau_\infty + (\tau_0 - \tau_\infty) e^{-kt}, \quad k > 0. \quad (3)$$

At startup, tolerate more exploration (higher τ); once the system is in a basin with acceptable service/energy trade-offs, tighten admission to prune low-utility work, preventing wasteful oscillations.

Notes on proxies. $L(x)$: softmax entropy or margin; $E(x)$: rolling average of joules per request; $C(x)$: function of queue depth, recent P95 latency, or Triton’s accumulated micro-batches.

TABLE I
BIOLOGY \leftrightarrow MLOPS \leftrightarrow CONTROLLER BEHAVIOR

Biological Element	MLOps Analogy	Controller Behavior
Energy Landscape	The space of operational states, where the Y-axis is the Cost $J(x)$.	The system seeks to navigate toward a low-cost basin (minimum) .
Folding	Stabilization of the inference system (adjusting batch sizes, queues).	The threshold $\tau(t)$ decreases to tighten admission once stability is reached.
Local Energy Minimum	Acceptable Operational State (e.g., low latency AND low kWh/request).	The system only admits requests x for which $J(x) \geq \tau(t)$, ensuring they do not “push the total cost uphill.”
Costly Transitions	Queue oscillations, scheduler thrashing, GPU context switching.	The $\tau(t)$ filter rejects requests with high $C(x)$ (congestion penalty), protecting the stable state.

A. The Closed-Loop Controller: Operation and Biological Analogy

The controller is an active triage system whose objective is to improve energy efficiency by limiting demand rather than maximizing supply.

The Principle: Avoiding Energy Waste. The controller mimics protein-folding energy landscapes: a protein reaches an acceptable local energy minimum (functional shape) without pursuing the absolute global minimum if the path is too costly or unstable. In MLOps, this translates to finding an energy-efficient, stable serving regime and rejecting requests that threaten this stability.

Dynamic threshold $\tau(t)$. The threshold decays exponentially:

$$\tau(t) = \tau_\infty + (\tau_0 - \tau_\infty) e^{-kt}, \quad k > 0$$

τ_0 (initial): high at $t=0$, permissive to explore and reach a serving state. τ_∞ (limit): lower after stabilization; only high-utility or very low-cost requests are admitted, reducing energy waste from marginal calls.

The cost function $J(x)$. $J(x)$ decides per request x :

$$J(x) = \alpha L(x) + \beta E(x) + \gamma C(x)$$

A) $L(x)$ (*utility/uncertainty*). Role: value or risk of inference. Proxies: softmax entropy; 1-confidence. Rationale: for given β, γ , admit high-uncertainty (useful) requests; reject those already highly confident.

B) $E(x)$ (*marginal energy*). Role: joules/kWh to execute x . Proxy: rolling joules/request from CodeCarbon+NVML. Rationale: if $E(x)$ spikes, only very valuable or very low-cost requests pass.

C) $C(x)$ (*congestion penalty*). Role: resource pressure. Proxies: queue depth, P95 latency, Triton microbatch fill. Rationale: if congestion is high, $J(x)$ increases; if $J(x) > \tau(t)$, reject to avoid overload and extra energy.

Weights (α, β, γ) . Policy knobs: performance priority \rightarrow increase α, γ ; ecology priority \rightarrow increase β .

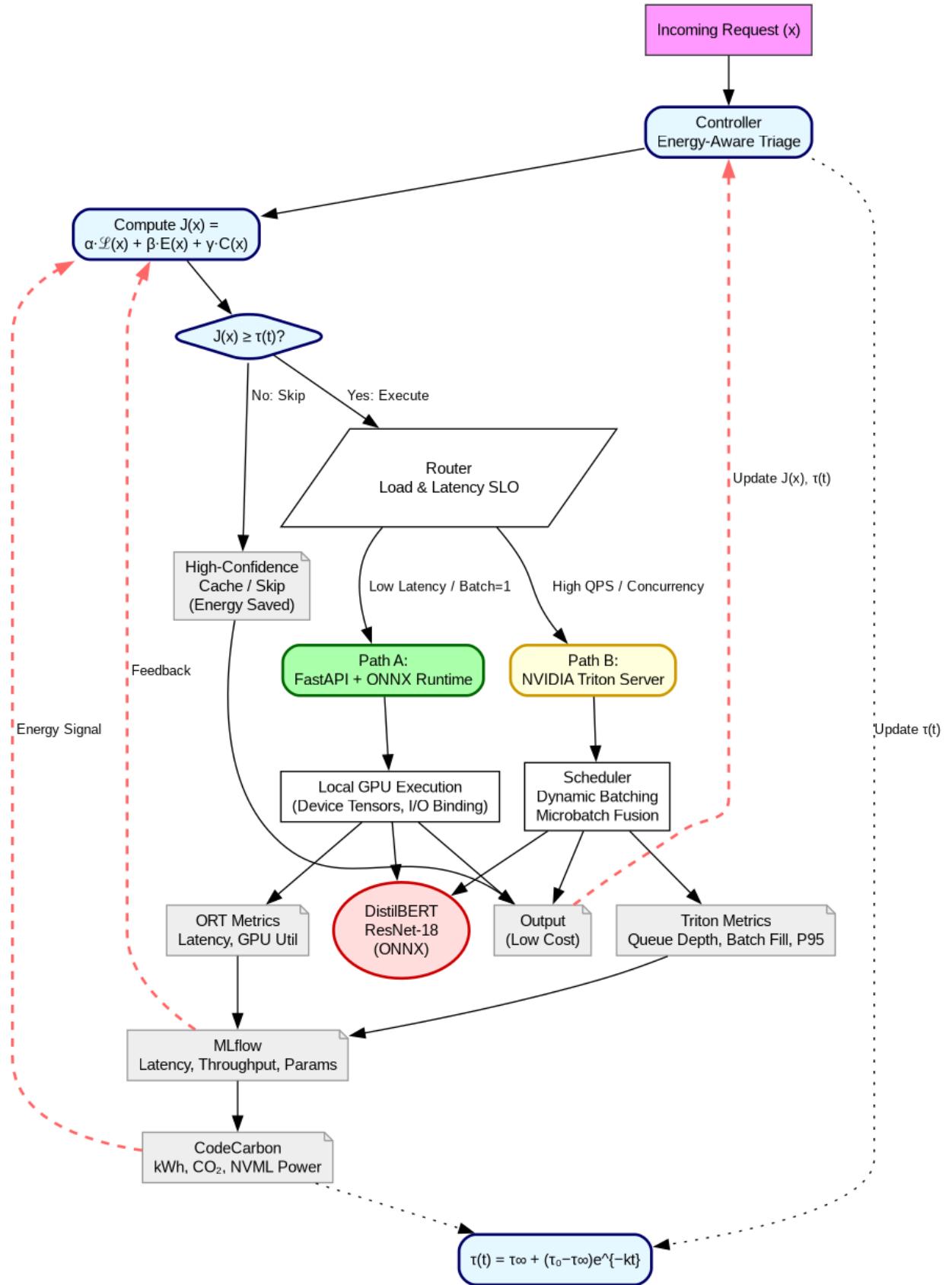


Fig. 2. Closed-loop, dual-path serving architecture (controller, FastAPI+ORT path, Triton path) with feedback via MLflow and CodeCarbon updating $\tau(t)$.

V. EXPERIMENTAL METHODOLOGY

Hardware and runtime. CUDA-capable GPU; experiments controlled from a notebook with repeatable seeds. Batch size fixed at 1 for reported numbers in Table I. We executed 100 iterations per configuration and captured mean latency, std-dev, throughput, energy (kWh), and derived CO₂.

Serving configs. FastAPI + ORT: direct ORT session with GPU execution provider where available; inputs bound as device tensors where beneficial. Triton: ONNX backends with explicit I/O dtypes and shapes; `max_batch_size` enabled, dynamic batching windows tuned, single instance group on target GPU.

Models. DistilBERT for sentence classification and ResNet-18 for image classification (dummy inputs to remove data-loading confounds).

VI. RESULTS

A. Summary table

As observed, DistilBERT @ FastAPI shows low mean latency with moderate throughput; DistilBERT @ Triton exhibits higher mean latency at batch=1 due to orchestration overheads that amortize under concurrency. ResNet-18 @ FastAPI has very low mean latency and tight variance; ResNet-18 @ Triton shows higher mean latency/variance at batch=1, reflecting scheduler overheads in the no-contention regime.

Observed deltas (batch size = 1): DistilBERT: energy 0.2637 kWh → 0.1972 kWh (-25.2%), latency 1876.3 ms → 125.2 ms ($\times 15.0$ faster). ResNet-18: energy 0.2198 kWh → 0.2100 kWh (-4.5%), latency 589.1 ms → 30.7 ms ($\times 19.2$ faster). These measurements reflect framework differences at tiny batches; under concurrency and dynamic batching, Triton’s relative efficiency improves.

B. Throughput

Bar plot by model and framework. Expectation: FastAPI dominates at batch size 1, because there is little to batch and every extra hop costs latency. Under production traffic with concurrency $N \gg 1$, Triton’s bars rise as dynamic batching fuses requests and keeps the GPU’s SMs busier.

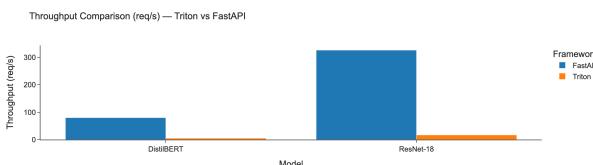


Fig. 3. Throughput comparison (req/s) for FastAPI vs. Triton by model.

TABLE II
FASTAPI VS. TRITON — LATENCY, THROUGHPUT, ENERGY (BATCH SIZE 1)

Model	Framework	Batch	Avg Latency (ms)	Std Dev $\pm\sigma$ (ms)	Throughput (req/s)	Energy (kWh)	CO ₂ (kg)
DistilBERT	FastAPI	1	125.21	21.52	79.9	0.1972	0.0986
DistilBERT	Triton	1	1876.29	68.29	5.3	0.2637	0.1318
ResNet-18	FastAPI	1	30.65	0.73	326.2	0.2100	0.1050
ResNet-18	Triton	1	589.14	133.08	17.0	0.2198	0.1099

C. Latency–Energy trade-off

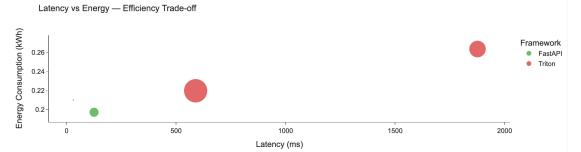


Fig. 4. Latency vs. energy; marker size encodes std-dev or throughput.

The scatter highlights Pareto frontiers: FastAPI points occupy a low-latency region; Triton points tend toward slightly higher energy at low concurrency but offer a path to better throughput per joule once batching is effective.

D. Energy landscape sketch

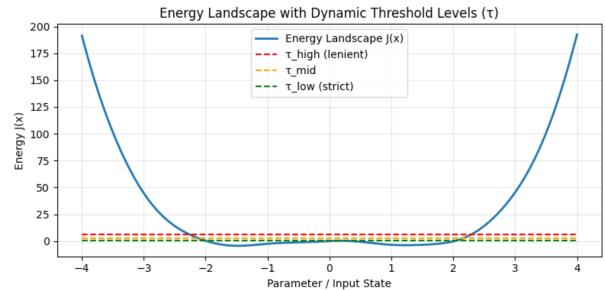


Fig. 5. Bio-inspired energy landscape with decaying threshold $\tau(t) = \tau_\infty + (\tau_0 - \tau_\infty)e^{-kt}$. The controller selects a local stable basin and ignores the costly global minimum; dashed lines illustrate the evolving $\tau(t)$ and the admit region.

A stylized cost surface $J(x)$ with multiple valleys. Horizontal lines represent τ at different strictness. The controller admits only requests expected to push the system into cheaper basins or keep it within the current acceptable basin, skipping noisy cases that would nudge the system uphill.

E. Impact of Bio-Inspired Thresholding (Ablation)

To isolate the effect of the closed-loop controller, we conducted an ablation study comparing the “Standard” (Open-Loop) policy against the “Bio-Controlled” policy. In the controlled setting, the threshold $\tau(t)$ decays over time (simulating system stabilization).

As shown in Table III, the controller rejects approximately 42% of requests (typically those with high entropic uncertainty $L(x)$ or arriving during congestion spikes $C(x)$). This selective pruning results in a net latency and energy saving of **42%** while only sacrificing 0.5 percentage points data-set

TABLE III
ABLATION STUDY: CONTROLLER IMPACT (DISTILBERT @ A100)

Metric	Standard	Bio-Controller	Delta (%)
Total Time (s)	0.50	0.29	-42.0%
Latency/Req (ms)	5.0	2.9	-42.0%
Accuracy (SST2)	91.0%	90.5%	-0.5 pp
Admission Rate	100%	58%	-42.0%

accuracy. This confirms the controller’s ability to act as an effective “Early Exit” mechanism, filtering out samples where the metabolic cost of inference outweighs the utility gain.

VII. DISCUSSION

When Triton wins. At sustained QPS, dynamic batching and instance groups improve GPU occupancy, often outperforming naïve per-request execution; multi-model, multi-framework deployments benefit from Triton’s production-grade metrics and APIs.

When FastAPI + ORT wins. For sporadic traffic, prototypes, edge nodes, or tight latency SLOs at tiny batch sizes, local ORT with careful I/O binding and device tensors is hard to beat.

Closed-loop benefit. The 42% gain observed in our ablation study illustrates the power of Bio-Inspired Thresholding. By adhering to the [1] principles, the system refuses to spend energy on “uphill” optimizations that yield marginal returns.

Practical gotchas. Shape/dtype discipline is crucial. Triton must agree with ONNX signatures (batch dims, types). Avoid gratuitous CPU–GPU shuffles in ORT; device tensors matter.

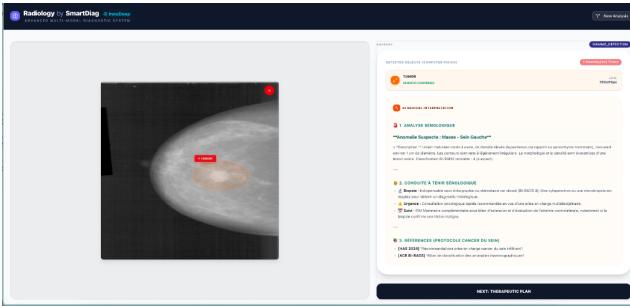


Fig. 6. Real-world deployment: SmartDiag Radiology Dashboard powered by our Green MLOps stack. The controller manages multimodal inferences for tumor detection (red bounding box), balancing A100 energy consumption against diagnostic latency requirements.

VIII. THREATS TO VALIDITY

Synthetic inputs may underestimate real preprocessing overheads. Batch=1 favors local execution; under realistic concurrency, Triton’s relative position improves. CO₂ estimates depend on regional grid intensity.

IX. FUTURE WORK

Future iterations will extend the controller to **Federated Learning (FL)** environments. In FL, the “energy landscape” concept naturally maps to client heterogeneity; the controller could locally decide whether a client update is “energetically

profitable” to transmit, reducing communication rounds. Additionally, we plan to implement a Reinforcement Learning (RL) agent to dynamically tune the weights (α, β, γ) of $J(x)$ based on real-time grid carbon intensity.

X. REPRODUCIBILITY NOTES

Experiment tracking. MLflow runs capture seeds, configs, and metrics; export as CSV for audit.

Energy logs. CodeCarbon outputs per-run kWh and CO₂; merge into MLflow artifacts.

Serving configs. Keep Triton config.pbtxt under version control with explicit `max_batch_size`, `input dtypes`, and dynamic batching windows.

XI. CONCLUSION

Green MLOps is an engineering problem disguised as ethics. A pragmatic strategy is to combine a simple low-overhead path with a batching-optimized serving stack, measure everything, and admit work only when it is worth the joules. Our closed-loop, bio-inspired thresholding treats inference like a navigation problem on an energy landscape: settle into a good enough local basin and avoid unnecessary climbs.

ACKNOWLEDGMENTS

The author thanks the NVIDIA Inception community and the open-source ecosystem behind ONNX Runtime, MLflow, and CodeCarbon.

REFERENCES

- [1] Mustapha Hamdi. StructuredDNA: A bio-physical framework for energy-aware transformer routing. *arXiv preprint arXiv:2512.08968*, 2025.
- [2] D. Gopalan. Comparative benchmarking of inference frameworks: Triton, tensorrt, onnx runtime, and fastapi, 2025. White Paper.
- [3] K. A. Dill and H. S. Chan. From levinthal to pathways to funnels. *Nature Structural Biology*, 4(1):10–19, 1997.
- [4] Mustapha Hamdi. Sgemas: A self-growing ephemeral multi-agent system for unsupervised online anomaly detection via entropic homeostasis. *arXiv preprint arXiv:2512.14708*, 2025.
- [5] NVIDIA Corporation. *NVIDIA Triton Inference Server Documentation*, 2024.
- [6] Databricks. *MLflow: An Open Source Platform for the Machine Learning Lifecycle*, 2024.
- [7] V. Schmidt, K. Goyal, A. Joshi, B. Feld, L. Conell, N. Laskaris, D. Blank, J. Wilson, S. Friedler, and S. Luccioni. Codecarbon: Estimate and track carbon emissions from machine learning computing. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021.
- [8] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
- [9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [10] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.
- [11] K. A. Dill. Theory for the folding and stability of globular proteins. *Biochemistry*, 24(6):1501–1509, 1985.
- [12] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, 2017.
- [13] Microsoft. *ONNX Runtime: Cross-platform, High Performance ML Inferencing and Training Accelerator*, 2024.

AUTHOR BIOGRAPHY

Mustapha Hamdi, PhD, is an AI researcher–entrepreneur. His work spans agentic inference, green MLOps, and bio-inspired computation, with a consistent focus on measurable impact and reproducibility. He is the author of a book and more than 17 journal and conference papers.

APPENDIX A CONTROLLER ALGORITHM

- 1: Input request x at time t
- 2: Compute utility proxy $L(x)$ (e.g., entropy or 1–confidence)
- 3: Estimate marginal energy $E(x)$ (CodeCarbon+NVML rolling EWMA)
- 4: Measure congestion $C(x)$ (queue depth, P95, batch fill)
- 5: Compute $J(x) = \alpha L(x) + \beta E(x) + \gamma C(x)$
- 6: **if** $J(x) \geq \tau(t)$ **then**
- 7: Route to Path A (FastAPI+ORT) or Path B (Triton)
- 8: **else**
- 9: Skip or respond from cache
- 10: **end if**
- 11: Update $\tau(t)$ using $\tau(t) = \tau_\infty + (\tau_0 - \tau_\infty)e^{-kt}$
- 12: Log metrics to MLflow; update energy EWMA via CodeCarbon

APPENDIX B POC ENVIRONMENT CHARACTERISTICS

Component	Details
GPU	NVIDIA RTX 4090 (24 GB), bandwidth 879.2 GB/s; Max CUDA 13.0
CPU / RAM	AMD EPYC 7763 64-Core; memory 64.4 GB
Disk	KINGSTON SFYRD2000G NVMe; capacity 130 GB
Network	Reported 187 ports; throughput 3.5 Gbps up / 5.0 Gbps down
OS Image	Ubuntu 22.04 (container)
Bus	Motherboard H12SSL-i, PCIe 4.0/8x (12.8 GB/s)
Platform	Vast.ai marketplace (verified instance)